(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 616 289 B1

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication and mention of the grant of the patent:
24.05.2000 Bulletin 2000/21

(51) Int. Cl.[7]: **G06F 17/30**

(21) Application number: **94301335.9**

(22) Date of filing: **24.02.1994**

(54) **System and method for interactively formulating queries**

Verfahren und System zum formulieren interaktiver Abfragen

Procédé et système pout formuler des requêtes interactivement

(84) Designated Contracting States:
DE FR GB

(30) Priority: **17.03.1993 US 32914**

(43) Date of publication of application:
21.09.1994 Bulletin 1994/38

(73) Proprietor:
International Business Machines Corporation
Armonk, N.Y. 10504 (US)

(72) Inventors:
- **Jacopi, Tom William**
  San Jose, California 95119 (US)
- **Payton, Brian Gerrit**
  San Jose, California 95130 (US)
- **Siwek, Howard Alexander**
  San Jose, California 95118 (US)

(74) Representative:
Williams, Julian David
IBM United Kingdom Limited,
Intellectual Property Department,
Hursley Park
Winchester, Hampshire SO21 2JN (GB)

(56) References cited:
EP-A- 0 491 517 **WO-A-92/16903**

- **IBM TECHNICAL DISCLOSURE BULLETIN., vol.35, no.4A, September 1992, NEW YORK US pages 101 - 102 'Dynamic, interactive Show SQL window'**

Note: Within nine months from the publication of the mention of the grant of the European patent, any person may give notice to the European Patent Office of opposition to the European patent granted. Notice of opposition shall be filed in a written reasoned statement. It shall not be deemed to have been filed until the opposition fee has been paid. (Art. 99(1) European Patent Convention).

EP 0 616 289 B1

Description

[0001] The present invention relates to computerized relational database informational retrieval systems, and more particularly relates to user interfaces for retrieving data from relational databases.

[0002] Computerized Database Management Systems (DBMSs) enable enterprises to efficiently manage and access data. DBMSs store data electronically in data storage devices such as direct access storage devices (DASD). Computer programs run by a computer system provide access to the database. The database user controls the computer processor and memory from a terminal using interaction devices such as a keyboard or a mouse.

[0003] The DBMS provides access to the database by logically controlling the storage and retrieval of the data. A relational DBMS logically stores the data organized into a plurality of tables consisting of a plurality of rows and columns. Columns contain data of the same kind. Rows contain different kinds of data about a single thing. The tables and columns are assigned names through which a user can identify tables, columns and rows for data retrieval.

[0004] Data is retrievable from a relational database by a variety of methods. A computer program can extract information from the database without any human user intervention. Additionally, database users can interact with the database using a query system program serving as an access program to the database system.

[0005] A common interface to a relational database is the Structured Query Language (SQL). SQL queries identify the database, the columns and rows for retrieval of the data. The columns are selected in the SQL SELECT statement. The rows are determined by the condition statements in the SQL SELECT statement WHERE clause. The condition statement identifies rows as a function of at least one column of at least one table. For example, an enterprise may have an employee database from which reports are generated from time to time, such as a report on the names of employees in a sales department who have been with the company for more than 10 years. An SQL statement would be used to retrieve the data from the employee database by retrieving all the employee names from a name column of a database table where the corresponding column in the same row for employee department has the value "sales" and the corresponding column in the same row for employment years has a value greater than 10. The SQL statement would look like:

```
SELECT EMP.NAME
        FROM EMP_TABLE
WHERE EMP.DEPT = "sales" and EMP.YEARS > 10.
```

[0006] The foregoing SQL statement is a simplistic example of a query statement that is used to retrieve data stored in a computerized database system. The queries can be quite complex involving many tables and many conditions placed on the row values in order to identify the rows to be retrieved.

[0007] There are a number of commercially available database query user interface programs which help the database user interactively construct queries for data retrieval. One such database interface program is the QMF product from the IBM Corporation. One feature of the QMF product which assists users in writing SQL queries is called Prompted Query. In Prompted Query, users can write conditions to restrict the answer set (rows of data) returned by the DBMS. These conditions are placed in the SQL WHERE clause. The conditions are comprised of predicates which are connected by AND and OR operators. Predicates are statements having variables that are substituted having a true or false value. For example, "EMP.DEPT=XYZ" and "EMP.YEARS>20" have true or false values depending on the value for the variables EMP.DEPT and EMP.YEARS.

[0008] One area which needs improvement in the Prompted Query system for writing the SQL condition statements is the limitations for specifying the precedence of the predicates. The condition statements can be very complicated with a number of predicates which need to be nested in parentheses in order to specify the precedence for the relationship between the predicates. For example, a query can try to retrieve all employees in a given department who have been there for ten years or employees in a different department who have been there for twenty years or any employee who has been there more than 30 years. The Boolean algebra statement for the condition can be written as: ((EMP.DEPT = XYZ) AND (EMP.YEARS > 10)) OR ((EMP.DEPT = ABC) AND (EMP.YEARS > 20)) OR (EMP.YEARS > 30). The parentheses help to determine the relationship of the predicates for satisfying the condition statement. Prompted query is limited by an inability to provide parentheses for nesting predicates.

[0009] When the number of predicates becomes large, and the degree of ordering of the predicates satisfying the queries is complicated, there is a need to indicate the order and relationship between predicates using parentheses. Moreover, it is difficult for the user to construct complicated queries using only the Boolean algebra statement text representation.

[0010] Many Boolean statements are logically equivalent; however some are more of an efficient representation of the condition statement. For example, a graph that can be represented by the Boolean algebra statements (P1 AND P4) OR (P3 AND P5) OR (P1 AND P2 AND P5), can also be represented by the Boolean algebra statement (P5 AND (P3 OR (P1 AND P2))) as well as (P1 AND (P4 OR (P2 AND P5)) OR (P3 AND P5). That is, all of the preceding Boolean statements are logically equivalent. However, the latter statement is

the most efficient representation with the least repetition of predicates. There is a need when translating between a Boolean representation and a graphical representation that consistency be maintained so that the translations are not confusing for the user.

[0011] There is also a need to better represent the flow of the conditions so that the user can more readily understand the results of the query.

[0012] EP 0 491 517 A4 discloses a system and method for graphically representing a query for retrieving data from a relational database, whereby logical operators are defined and linked to predicates using a tree structure format.

[0013] In general, there is a need for allowing users to access databases without having to know the complicated syntax of SQL queries.

[0014] In accordance with the present invention, there is now provided a method for interactively formulating queries for retrieving data stored in tables having columns and rows in a computerized database system having a central processing unit (CPU), memory, a display device, and a user interaction device, the method comprising the steps of: receiving input identifying at least one table containing the query data; receiving input identifying at least one column of the identified table containing the query data; receiving input identifying at least one row in the identified table containing the query data using a condition statement and characterised in that said condition statement is interactively formulated by the steps of: (a) displaying a graphical flow representation comprising at least one flowline comprising at least one arc, a predicate assigned to each arc, and a plurality of nodes, each arc positioned between a From_Node node and a To_Node node, wherein predicates on the same flowline are combined through a logical AND operation, and predicates on different flowlines between the same nodes are combined through a logical OR operation; (b) receiving input specifying a position on a flowline of the graphical flow representation for insertion of a predicate; (c) receiving input of the predicate; (d) displaying the graphical flow representation with the predicate inserted in the specified position; (e) converting the graphical flow representation into a corresponding Boolean algebra statement; and (f) displaying the Boolean algebra statement.

[0015] Viewing the present invention from another aspect, there is now provided a method for use in a computer system for converting a Boolean algebra statement representation to a graphical representation comprising the steps of: a) arranging the Boolean statement into a parse tree wherein each element in the parse tree has a tree From_Node and a tree To_Node and each element is a leaf or a non-leaf, and each non-leaf has a left child and a right child, wherein a leaf is a condition predicate and a non-leaf is an OR operator or an AND operator; and (b) processing each element of the tree in preorder traversal to assign each condition predicate as a graph arc defined as connecting a graph From_Node to a graph To_Node comprising the steps of: assigning the root element From_Node and To_Node as the Beginning_Node and Ending_Node of the graph; for an OR operator element, assigning an OR operator element From_Node and To_Node as the From_Node and To_Node of both of the OR operator element children; for an AND operator element, creating a New_Node, assigning the AND operator element From_Node as the From_Node of the left hand child of the AND operator element, the AND operator element To_Node as the To_Node of the right hand child of the AND operator element, assigning the New_Node as the To_Node of the left hand child and the from node of the right hand child of the AND operator element.

[0016] Viewing the present invention from yet another aspect, there is now provided a method for use in a computer system for converting a graphical representation comprising a plurality of nodes linked by arcs into a Boolean algebra statement representation, comprising the steps of: (a) when two arcs have the same To_Nodes and From_Nodes, combining the arcs into one arc with the conditions represented by the original arcs linked by a logical OR operand; (b) when two arcs are in a series with a Third_Node in between having the same two arcs going into and out of it, combining the arcs into one arc with the conditions represented by each original arc linked by an AND operator; (c) pushing onto a stack a graph path; (d) popping a path from the stack for further processing; (e) processing the path to produce a Boolean algebra statement segment while identifying any new paths; (f) pushing a new path onto the stack; and (g) repeating steps (d) through (f) until the stack is empty.

[0017] In a preferred embodiment of the present invention, there is provided computerized database system having a central processing unit (CPU), memory, a display device, and a user interaction device, a method is provided for retrieving data. At least one table, column and row are interactively identified containing query data. The rows are identified by means of a condition statement. The condition statement is interactively formulated by the following procedure. At least one flowline is displayed and input is received from the user specifying a position on the flowline for insertion of a predicate and the predicate. The flowline is displayed with the predicate inserted in the specified location. The flowline is converted to a corresponding Boolean algebra statement which is displayed for the user. Additionally, the user can add a predicate to the Boolean algebra condition statement. The Boolean algebra statement is converted to the corresponding graphical representation and displayed for the user. The predicates are displayed on the flowlines in between nodes. Predicates on the same flowline are combined through a logical AND operation. Predicates on different flowlines between the same nodes are combined through a logical OR operation.

[0018] The Boolean algebra text representation is

converted to the graphical representation by first arranging the Boolean statement into a parse tree. Every element of the tree is processed in preordered traversal where each element in the tree is either a predicate, an OR operator, or an AND operator and predicates are leaves and operators are non-leaves having two children. Each element has a From_Node and To_Node associated with it corresponding to the nodes of the directed graph representation. The root of the parse tree has its From_Node and To_Node assigned the value of the beginning node and ending node of the resulting graph. The children of OR operator elements are assigned the From_Node and To_Node of the OR operator element. For elements that are AND operators, a new node is created. The From_Node of the left child is assigned the From_Node of the AND operator element. The To_Node of the left child and the From_Node of the right child are assigned the new node. The To_Node of the right child is assigned the To_Node of the AND operator element. For predicate elements, an arc is created connecting the From_Node to the To_Node of the element and assigned the predicate.

[0019] The graphical flow representation graph is converted to the Boolean algebra statement representation by first compressing the graph and then assigning the Boolean statement values.

[0020] The graph is compressed from the inside out down to the fewest nodes and arcs as possible by combining two arcs and replacing the two with one arc with the resulting Boolean algebra statement as the predicate. When two arcs of the graphical flow representation graph have the same To_Node and From_Node, the arcs are combined into one arc with the condition predicates represented by the original arc connected by a logical OR operation which is assigned to the resulting arc. When two arcs are in a series with a node in between and the node only has the two arcs going into and out of it, then the arcs are combined into one arc with the condition predicates represented by each original arc connected by a logical AND operation which is assigned to the resulting arc.

[0021] The assignment algorithm identifies unique paths through the graph from the Starting_Node to the Ending_Node. The Boolean algebra statement is created while the compressed graph is traversed. Each path through the graph is comprised of a plurality of predicates combined through a logical AND operation. The Boolean algebra statement text is constructed by determining whether an OR or an AND operator needs to be added to the text statement to correspond to the graphical representation. Also, the predicate and beginning and ending parentheses are added to the text statement. An "OR" is inserted into the Boolean text when a new path has been identified. A stack is used to store each new path through the graph. When the path does not go to the end node of the graph an "AND" is inserted into the Boolean text. After all arcs out of a node have been processed, a new path is popped from the stack. When the stack is empty, a corresponding efficient Boolean algebra statement for the graphical condition is complete.

[0022] It is an object of the invention to provide a process by which a database user can formulate queries using a graphical representation to illustrate the flow of the data through condition filters.

[0023] It is a further object of the invention to provide a process for consistently translating a condition statement between a graphical flow representation and a textual Boolean algebra statement.

[0024] A preferred embodiment of the present invention will now be described with reference to the accompanying drawings, in which:

Fig. 1 is a schematic drawing of the computer system;

Fig. 2 is an overview of the query flow process;

Fig. 3 is a flow chart of the query condition modification process;

Fig. 4 is an initial row selection window;

Fig. 5 is a graphical condition window illustrating the addition of a new path;

Fig. 6 is the graphical window condition of Fig. 5 illustrating the addition of the new path and a NOT operation;

Fig. 7 is a row selection window with three row predicates;

Fig. 8 is the row selection window of Fig. 7 with the predicates in a stacked format;

Fig. 9 is a group selection window;

Fig. 10 is a sort window with sort columns;

Fig. 11 is an overview of the query flow process with UNION;

Fig. 12 is a flowchart of the compression algorithm;

Fig. 13 is pseudocode of the compression algorithm of Fig. 12;

Fig. 14 is an example of an initial graph and representation;

Fig. 15 is the graph of Fig. 14 after arcs have been combined using the compression algorithm of Figs. 12 and 13;

Fig. 16 is a flowchart of the assignment algorithm;

Fig. 17 is pseudocode of the assignment algorithm of Fig. 16;

Fig. 18 is a fully compressed graph initialized for the assignment algorithm;

Fig. 19 is the flowchart of the Boolean algebra to directed graph algorithm;

Fig. 20 is pseudocode of the Boolean algebra to directed graph algorithm of Fig. 19;

Fig. 21 is an initial parse tree; and

Fig. 22 is the final arc list for the resulting directed graph from the initial parse tree of Fig. 21 using the algorithm of Figs. 19 and 20.

[0025] The query flow process conceptualizes the query formulation process in terms of the flow of data through filters which limit what data is to be retrieved.

[0026] The user interface for the query formulation process used for retrieving data from a database is implemented in connection with the data processing apparatus shown in Fig. 1. The apparatus comprises a central processing unit (CPU) 30, random access memory 32, input/output port 34, and nonvolatile storage 36 such as disk storage or read only memory (ROM), all connected to a common bus structure 38. Control circuitry 40 performs housekeeping operations, such as providing appropriate clock signals and controlling the operation of the bus. An adaptor 42 is used to interface to other components such as a keyboard 44, a visual display unit 46, and a mouse 48. The general purpose data processor shown in Fig. 1 is used to perform the invention under program control.

[0027] Referring to Fig. 2, the formulation of the query is displayed for the database user as broken into the sections of table selection, row selection, group selection and sorting of the data to be retrieved. These parts are navigated through by the user to provide the flow of the data through the query.

[0028] A window 50 is displayed for the user on the display device with icons 52 representing filters that restrict the flow of data through the query. The user selects an icon to elaborate on a stage of the query formulation process. In that way, the SQL query window is a window in which the user can write any SQL query. Once the SQL query is formulated, it is passed to the database to retrieve the data.

[0029] The full SQL statement defines the database tables and columns that form the source of raw data and the conditions that define those rows of interest from the defined database tables. The first icon represents table and column selection which corresponds to the segment of the SQL statement preceded by the word SELECT. The user selects the table icon 55 in order to follow a set of procedures for interactively identifying the tables and columns which provide the query data.

[0030] After the tables and columns are selected, the user next formulates the row conditions for specifying the data to be retrieved. The segment of the simple SQL select statement preceded by the word WHERE contains the Boolean predicates of the a row condition for the query data.

[0031] The environment for the creation and manipulation of Boolean predicates as used in SQL for restricting row selection can be represented as a directed graph. Referring to Fig. 3, the present invention provides a representation to show the relationship between predicate conditions in a SQL statement WHERE clause on a display screen in two formats, textual and graphical. The user can initially enter a graphical condition 56 by the process discussed below. The system converts the directed graph representation into the corresponding Boolean algebra statement 58 by a process that will be explained below. The user then has the option of directly changing the Boolean algebra statement or the graphical representation 60. When either representation is changed by adding or rearranging predicates, both representations are updated. The user also can initially input a Boolean algebra statement 62 and the system converts the statement into the corresponding graphical representation 64.

[0032] The user formulates row conditions by selecting the row condition icon (66 in Fig. 2). Referring to Fig. 4, the initial window display 70 for the row conditions, when no row conditions are present, is an empty predicate box 72 on a flowline 74 between two nodes 76.

[0033] The row conditions are graphically displayed in a flowline directed graph format, indicating that data flows through the condition predicates which act as filters. The flowlines intuitively define AND and OR operations for predicates, while also adding order-of-precedence function to the condition. The predicates are placed in boxes on an arc with connectors or nodes shown as small squares between the boxes on the flowlines. The connectors are used for starting and ending points when adding new arcs.

[0034] The graphical representation illustrates the condition statement as a domain considered to flow through the set of condition predicates, which act as filters that only let through the domain elements that satisfy the applicable criteria. An AND operation is signified by placing two filters sequentially so any domain element must pass through both filters. An OR operation is shown by creating parallel paths separately containing the filters. The primitives can be combined to produce complex conditions.

[0035] In Boolean algebra statements, a user must look at the statement string and match up parentheses to determine the level of nesting for each condition. With the graphical representation, the nesting of predicates

is illustrated in a concise manner.

[0036] Referring to Fig. 5, the graphical condition window 80 is divided into three areas: a graph area 82, a text area 84, and a condition area 86. The graph area is where the graph representing the conditions is shown. The text area is the Boolean algebra statement equivalent of the graph. And the condition area contains a list of all the actual predicates.

[0037] The graph area contains three elements: at least one or more flowlines 88, nodes 90, and filters 92. The flowlines define the possible paths for domain elements to take as they work their way through the graph. At the flowline coming in from the left hand side, all domain elements are available. The flowline exiting the right hand side has the domain elements that have successfully traversed the graph. The boxes with condition predicate text in them are filters. The filters let only those domain elements that satisfy the condition predicate inside of them pass through. The boxes are on arcs of a flowline between a pair of nodes. The nodes on either side of the condition predicate window on the condition predicate squares serve as starting and ending locations for creating new flowlines.

[0038] The user can modify the set of conditions two ways, either using the text area or the graph area. The types of modifications that can be done include inserting, deleting or updating any part of the set of conditions. Modifications using the text area involves overtyping the Boolean statement string. The changes to the Boolean string are displayed on the Boolean string, and also in the graph area. Modifications using the graph area allow the user to add new filters and modify existing filters. New filters can be added as a new flowline connecting existing nodes which is equivalent to adding a logical OR operation. Additionally, new filters can be inserted into an existing flowline which is equivalent to adding an AND operation.

[0039] A predicate can be combined with another predicate through a logical OR operation by adding a new flowline to the graph between two existing nodes. As shown in Fig. 5, the user places the mouse pointer 96 on a node 97 and presses the selection button. While holding the selection button down, the mouse is moved off the node. A line 98 appears from the original node to the mouse pointer, indicating a new path that is being drawn. Eventually the mouse is moved to another node and the selection button is released. In this way, a path can be drawn between two nodes to create the new flowline.

[0040] Not all nodes are valid for the ending of a new path depending on the starting node. That is, no new path may be created that results in a circular path existing in the graph. A circular path is one in which it is possible to start at a node, traverse the graph and eventually end up back at the same node. When the mouse is over an invalid node, the user is notified that an invalid flowline would be created.

[0041] After the new line is completed, the user is prompted to enter the predicate text assigned to that arc. The predicate text is placed inside the filter and the new graph is displayed for the user. Referring to Fig. 6, the line 98 was drawn to the node 99 following the predicate Y<5 and the predicate Z=20 was added to the predicate filter box 100.

[0042] Predicates can be combined through a logical AND operation in the graphical representation by inserting a filter into an existing flowline. The user places the mouse over the position on the flowline where the new filter is to be entered and double clicks the mouse selector button. A new filter is inserted at that location, and any new nodes that are required by the new filter are inserted. The user is then prompted for the predicate text to fill in the filter.

[0043] Referring to Fig. 6, the flowline containing predicates X>100, C=5, and Y<5 are all combined using a logical AND operation. When the predicate X>100 was added to the flowline 102, prior to the insertion of the predicate Z=20, the mouse cursor was positioned over the node 103 to the left of the predicate B=4, a new node 97 and predicate filter box 104 were inserted into the flowline 102 containing predicates B=4 and Y<5, and the user input the predicate value X>100 in the predicate box 104.

[0044] There are other methods for interactively inserting predicates and flowlines that are well known to those skilled in the art.

[0045] Referring to Fig. 5, the Boolean algebra statement equivalent of the graphical flow representation is shown in the SQL clause box 84. After the insertion of the predicate Z=20 in the selected location in the graphical flow representation, the equivalent Boolean algebra statement that will be displayed is Z=20 OR (X>100 AND (A=3 OR ((B=4 OR C=5) AND Y<5))) (partially shown in Fig. 6).

[0046] Predicates can be modified by selecting them and then selecting the modification functions from a pop-up menu window. A predicate is selected by drawing a box around the desired filter using a mouse or similar user interaction device. More than one predicate filter can be selected at one time by drawing a box around all the desired filters. The selected predicate filters can be edited where the predicate text in the filter box is changed using a separate window for filling in an edited predicate text. The filter can also be deleted, at which time the flowline arc for the filter is also removed. When the deletion of a filter results in the two nodes that used to be connected by the flowline arc becoming unconnected by the deletion of the filter, the nodes are merged into one node.

[0047] Referring to Fig. 6, the conditions on the flowline can also be negated by drawing a box 122 around the predicate filter or filters to be negated. To add a box to indicate negation, a mouse or other user interaction device is used to select one or more condition predicates. Next, a negate option is selected from a menu. After the insertion of the NOT operation to the

selected section 122 of the flow graph, the resulting Boolean algebra statement is Z=20 OR (X>100 AND NOT (A=3 OR ((B=4 OR C=5) AND Y<5))).

[0048] Only predicate filters that form a valid subgraph may be negated together. It is a straightforward process to determine a valid subgraph based on the flowlines entering and leaving the set of selected predicate filters. Only one logical flowline can enter the set of selected filters from the outside non-selected filters, and only one logical flowline can leave the set of filters. One logical flowline is defined as either one actual flowline entering the set or two or more flowlines entering the set immediately originating from the same node. One logical flowline leaving the set is defined in much the same way. It is either one actual flowline or all flowlines immediately ending at the same node.

[0049] In the example shown in Fig. 6, the filter predicates A=3, B=4, C=5 and Y<5 130 are selected, three flowline arcs 131-133 enter the set of selected filters, one each to filters A=3, B=4 and C=5. However, this is one logical flowline since they all start from the same node 103. Also, all flowlines leaving the selected filters end at the same node 99. However, the filter Z=20 could not be included in the NOT operation since the set would then have four flowline arcs entering, where three of the flowlines originate from one node 103 while the fourth flowline enters from a different node 97. Since the flowlines originate from separate nodes, this is not a valid subgraph to negate. The negate choice is therefore unselectable on the menu.

[0050] Another example of a row condition formulation is illustrated in Fig. 7 where a user wants to restrict the rows of the query to all managers or commissioned employees whose total earnings are above $15,000. Initially, one predicate box appears in the row selection window. The user inputs the row condition predicate which is placed in the predicate box. The first predicate in the example is the restriction on the data to all employees who earn more than $15,000 140. A second predicate is then added to restrict the group to employees who earn a commission 142. Following the procedure for insertion of a new predicate as part of an AND operation, a second predicate box connector and prompt panel are provided for the user to enter the predicate for the new filter predicate which in this example is "COMM not NULL" 144. In order to add the predicate of the job being a manager 145, a new flowline 146 is created.

[0051] Referring to Fig. 8, the user may view the flow diagram in a stacked format 147 which stacks the predicates 140, 142 that are connected by the AND operator.

[0052] The condition formulation process can also be used for GROUP conditions. Once the user provides input to the system that forces the generation of groups, such as placing a summary function as part of the column selection process, then a group conditions icon (see 148 in Fig. 2) appears along with additional text under the row selection icon. For example, when the user selects the columns, department, job, and average salary from a table, the top level query window (Fig. 2) shows the flow of data as being from table selection, into row selection for groups, into group selection, into sorting.

[0053] Inside the row selection window, the grouping column information is shown on the right hand side of the window. Nothing in the grouping column section can be directly updated. To change the grouping columns, the user must go back and select or delete columns included in the query. Selecting the group selection icon causes a window (see Fig. 9) to be displayed showing a flowchart condition format. The left hand side of the group selection window 120 has information about which overall aggregate functions are computed. The list displays for the user what functions are being evaluated for the query and helps in the continuity of the flow from the row selection window.

[0054] The process for formulating conditions for groups is the same as for any conditions except that the group selection prompt panel has information about summary functions.

[0055] Following the selection of the table and columns for the query data and the formulation of the condition for the row or group selection, the next step in the query formulation process is the sorting of the query data. The sorting function window initially displays only a flowline and connectors. The user adds a sort column by placing the mouse cursor over the connector and pressing the mouse button. The user then drags the connector either above the line for ascending order or below it for descending order and releases the mouse button. A sort box appears on the selected side of the line, along with a list box that has all the selectable columns for sorting. After the user selects a column from the list box, it is displayed in the sort box.

[0056] Referring to Fig. 10, the user has selected sorting first by department name in ascending order and then by employee name in descending order. After department name was selected from a list box of columns and placed in the sort box 160, additional connectors 162 were displayed so that additional sort boxes could be created on either side of the existing sort box 164. The connector to the right of department name was selected and dragged downwards for the employee name sort. The order of precedence for sorting reads from left to right. Any sorts on the left of any other sort gets applied before the right hand sorts, so that the sort by descending employee name is less significant than the department name sort.

[0057] The formulation of queries involving the UNION operation between two or more sub-queries uses an initial query flow window shown in Fig. 11. The selected rows of data are combined by a UNION function 172 and then sorted 174. The UNION operation is selected from an action bar pull-down menu in the graphical query overview window. When the UNION

operation is selected, the query flow diagram as in Fig. 11 is displayed as part of the overview window. The user enters the table, column, and row selections for the queries by expanding the icons to specify query details. After the tables and columns are selected for a second query, the selected columns are verified against the first query to make sure the number of columns and their data types match, which is required for SQL UNION operation. After the row selections are made for each query, the query data are combined by a UNION operation.

[0058] The directed graph of the graphical flow representation for the row conditions is displayed as a Boolean algebra statement so that the user can view the row conditions in both formats for greater understanding. The directed graph is converted to a Boolean algebra statement using the algorithm in Figs. 13 and 17. The Boolean statement is converted to a directed graph using the algorithm in Fig. 20.

[0059] In general, a graph is a set of one or more nodes connected by zero or more arcs. Pictorially, nodes can be represented by squares, while arcs can be represented by lines connecting the squares. The identification for the nodes can be placed inside the squares. Arcs can be identified by a name placed by the line or by referring to the two nodes that the arc connects. A graph can be defined as either a list of nodes or a list of arcs.

[0060] A directed graph is a subset of a graph where the arcs indicate direction. Pictorially directed graphs are shown by means of arrows on the arcs. In the list format, directed graph arcs have From_Nodes and To_Nodes listed. A further subset of directed graphs are ones that have only one starting node and one ending node and no circular path.

[0061] It is possible to have a directed graph that has no Boolean condition that is exactly equivalent. However, it is always possible to generate a Boolean condition that is logically equivalent.

[0062] In this invention, the graphical flow representations are directed graphs with no circular paths. The actual conditions are represented by the arcs, whereas the nodes are placed according to the structure of the AND and OR statements in the conditions.

[0063] This invention converts the directed acyclic graph of the query flow representation to the corresponding Boolean algebra statement by first using a compression algorithm as shown in Figs. 12 and 13. The graphs are compressed from the inside out, down to as few nodes and arcs as possible. The compression algorithm always tries to combine two arcs and replace it with one arc. When two arcs have the same To_Nodes and From_Nodes 180, the arcs are combined into one arc, with the condition represented by each original arc connected by a logical OR operation 181. When two arcs are in a series with a third node in between, and the third node only has those two arcs going into and out of it 182, then the arcs are combined into one arc with the condition represented by each original arc connected by a logical AND operation and the intervening third node is eliminated 183.

[0064] The goal is to compress the graph down to only two nodes and one arc, which is a completely reduced graph. However, in some cases, the graph cannot be completely compressed. After the graph is compressed to its minimum number of nodes and arcs, the graph is converted to a Boolean algebra statement using an expansion and assignment algorithm.

[0065] The compression algorithm can be further explained using the simple graph shown in Fig. 14. The graph 184 consists of four nodes (A, B, C and D) 185 and five arcs (1 through 5 with predicates P1 through P5, respectively, assigned to each arc) 186. For each node in the graph, the following information is stored in a table 187: the number of arcs going into the node 188, the number arcs going out from the node 189, the list of the arcs going into the node 190, and the list of arcs going out from the node 191.

[0066] During the processing shown in Figs. 12 and 13, the node table (185 in Fig. 14) is processed in a loop 194, with each iteration trying to combine arcs. The loop is terminated when one complete pass through the node array results in no arcs being combined 196. Where two arcs go out of the same first node and into the same second node 180, the predicate values are combined with an OR operator and assigned to the first arc, and references to the second predicate are removed 181. For nodes where only one arc goes into the node and one arc comes out of the node 182, the predicate values are combined by an AND operation and assigned to the arc going into the node, and that arc is assigned to go into the node of the second arc 183.

[0067] In processing the example graph shown in Fig. 14 (according to the algorithm of Figs. 12 and 13), the first iteration skips nodes A and B since neither meet the criteria for compression. However, since node C has more than one arc going into it (197 in Fig. 13), the list of arcs going into it (arcs 1, 2 and 5) are each examined 181. Since arcs 1 and 2 both go out from the same node the predicates are connected by an OR operator and assigned to arc 1. All references to arc 2 are deleted. Next, since node D has an in arc count of 1 and out arc count of 1, it is removed. The predicates of the two arcs going into and out of node D are combined by an AND operator and assigned to the first arc. The first arc To_Node is updated with the To_Node of the second arc and the second arc is deleted.

[0068] Since changes were made during the last iteration or pass (arcs 1, 2 and 4, 5 combined) another pass of the node array is performed. At the end of the first pass, Node A has zero arcs going into it and two arcs (arcs 1 and 4) going out, Node B has one arc (arc 3) going into it and zero going out of it, Node C has two arcs (arcs 1 and 5) going into it and one arc (arc 3) going out of it. Node D has been deleted. Arc 1 has the predicate (P1) OR (P2) assigned to it, arc 2 has been

deleted, arc 3 has the predicate P3 assigned to it, and arc 4 has the predicate (P4) AND (P5) assigned to it.

[0069] On the second pass, Node C is examined because it has an in arc count greater than one. Examining the list of arcs going into Node C shows that both arcs 1 and 4 have the same From_Node, Node A. Therefore, the two arcs, 1 and 4, are combined with an OR operation. At the end of the second pass, Node A has zero arcs going into it and one arc (arc 1) going out, Node B has one arc (arc 3) going into it and zero arcs going out of it, Node C has one arc (arc 1) going into it and one arc (arc 3) going out of it. Arc 1 has the predicate ((P1) OR (P2)) OR ((P4) AND (P5)) assigned to it, arc 3 has the predicate P3 assigned to it, and arc 4 has been deleted.

[0070] On the next loop of the node array, Node C has an in arc count of one and an out arc count of one. The arcs going into and out of Node C are combined by an AND operator. The result of this compression operation is a reduced graph down to the minimum possible graph of two nodes and one arc, 200 in Fig. 15. There are two nodes A and B and the predicate assigned to the arc AB is ((P1) OR (P2)) OR ((P4) AND (P5)) AND (C5).

[0071] The foregoing example is not always the situation. The graph 201 shown in Fig. 18 is an example of a graph where the node and arc arrays cannot be compressed any further, but it has not been reduced to two nodes and one arc.

[0072] The assignment algorithm in Figs. 16 and 17 assigns the corresponding Boolean algebra statement regardless of whether the compressed graph is fully reduced.

[0073] The assignment or expansion algorithm tries to find each unique path through the graph from the starting node to the ending node. The graph itself is not manipulated any further. Instead, the Boolean text string is created while the graph is being traversed. Since only one path can be traversed at a time, a stack is used to keep track of each new path through the graph. When the stack is empty, no more paths are to be processed; and the Boolean condition is complete. The expansion algorithm traverses each path through the graph building the Boolean algebra statement string along the way. The algorithm inserts part of the text into the main string a piece at a time. The list of places that need to have text inserted is kept on the stack.

[0074] When a graph has been reduced down to two nodes and one arc such as the compressed graph in Fig. 15, the expansion algorithm has little to do since only one possible path exists from start to end.

[0075] Referring to Figs. 16 and 17, the first step is the initialization of the variables for the expansion algorithm 205. Two of the variables are the start node (the starting node of the graph) and the end node (the ending node of the graph). The start and end nodes can be determined from the in and out arrays used in the compression algorithm. The start node has an in-arc-count of zero and an out-arc-count of greater than zero. The end node has an in-arc-count greater than zero and an out-arc-count equal to zero. Another variable the Boolean Text, is the text string that holds the Boolean algebra statement. The text variable has an initial value of null.

[0076] A stack holds the pair of items, a node value and a text position pointer. The node represents where in the graph the current process is active, while the position pointer is where the insertion into the text string is occurring. The start node and a place value indicating the start of the text is initially pushed onto the stack.

[0077] After initialization is complete, the graph is continually processing nodes stored in the stack until the stack is empty. The top values are popped from the stack and placed into the variable's current node and current place and the arc count is set to zero 207. For each arc in the graph, if the From_Node for the arc is the same as the current node, then the arc count is incremented by one 208. If the arc count is greater than one, then the text "OR" is inserted into the Boolean text at the location defined by the place variable, and the place variable is incremented to point after the inserted text 209. When the To_Node for the arc is not the same as the end node (the arc does not go to the end node), then a begin parenthesis "(" is placed into the Boolean text at the location defined by the place variable and the current place is incremented so it points after the inserted text 210. The predicate text assigned to the arc is inserted into the Boolean text at the location defined by place, and the place is incremented to after the insertion 211. If the To_Node for the arc is not the same as the end node, then the text AND and a begin parenthesis is placed into the Boolean text at the location defined by place, and the place variable is incremented to after that text position; the arc's To_Node and the place variable are pushed onto the stack; and the text of two end parentheses "))" is also added to the text at the location of the place variable and the place counter variable is incremented 212.

[0078] The expansion algorithm can be further explained using the example in Figure 18 showing a fully compressed graph 201. In the example, there are four nodes, A, B, C, and D 213 and five arcs 1 through 5 with predicates P1 through P5, respectively, assigned to each arc 214. As listed in the node array 215 (and shown in the graph 201), Arc 1 goes from Node A to Node C, Arc 2 goes from Node C to Node D, Arc 3 goes from Node A to Node D, Arc 4 goes from Node C to Node B, Arc 5 goes from Node D to Node B. From looking at the arcs into nodes and arcs going out of the nodes it can be determined that node A is the starting node and node B is the ending node. The stack 216 after initialization holds node A and the value for the position variable (also called "Place") of 1.

[0079] Node A and the Place value of 1 are popped off the stack. The arc list is searched for any arcs that have Node A as the From_Node and the first arc, Arc 1,

is identified. The Arc count variable is incremented one. Since the To_Node of Arc 1, Node C, is not the end node, Node B, a left parenthesis is added to the text at the position of Place (210 in Fig. 16). Next, the actual predicate text associated with Arc 1, P1, is placed into the string text variable at the position of Place (211 in Fig. 16). Next the word AND is added along with the left parenthesis. The To_node of arc 1, Node C, and the current value of Place (right after the AND and left parenthesis), is pushed onto the stack. Finally, two right parentheses are placed into the string and the Place variable incremented to that position (212 in Fig. 16). At the end of the first iteration, the Boolean text has the value of "(P1 AND ( ))" and the current value of Place is following the right most end parenthesis; also, the Node C and the Place value following the "AND (" were pushed onto the stack.

[0080] The process continues identifying more arcs having node A as the From_Node (220 in Fig. 16) and Arc 3 (P3) is identified. The arc count variable is incremented to two (indicating that the current node has two arcs out of it, which means that there is an OR operation between the predicates on the arcs. The text "OR" is placed in the string at the current place (after the right most parenthesis). Since the To_Node of Arc 3, Node D, is not the end node, the same thing is done as for the previous arc, arc 1. That is, at the end of this iteration the Boolean text is "(P1 AND ()) OR (P3 AND ())" (P3 being the predicate assigned to Node D) where the Place variable is at the end of the right most parenthesis and the stack has Node D and the position pointer (Place) following the string "P3 AND (" pushed onto it in addition to the Node C and Place value following "P1 AND (".

[0081] Since no other arcs have the current node, Node A, as the From_Node, a new node is popped from the stack, Node D, and a new Place pointer value, following the text "P3 and (", are assigned as the current Node and Place values. There is only one arc with Node D as its the From_Node, Arc 5. Since the To_Node of Arc 5 is the end node, Node B, all that is done is the addition of the predicate text assigned to Arc 5 to the text variable at the location of the current Place. At the end of this iteration, the text string is "(P1 AND ()) OR (P3 AND (P5))" and the stack still has Node C and the Place value following "P1 AND (".

[0082] The stack is popped again to get the value for the current Node of Node C and the place value following the text segment "P1 AND (". The first arc found with the Node C as its From_Node is Arc 2. Arc 2 has Node D as its To_Node (which is not the end node). The arc count for the current Node has the value of one and the predicate P2 is assigned to arc 2. The process is similar as before. The resulting text string is "(P1 AND ((P2 AND ( )))) OR (P3 AND P5)) with the Place pointer pointing after "( ))". The stack contains Node D and the Place value pointing to the position after "P2 AND (". The next arc with Node C as its From_Node is Arc 4 with the predicate P4 assigned to it. The arc count is incremented to two. The To_Node of Arc 4 is the end node, Node B. The resulting text string after this iteration is "P1 AND ((P2 AND ( )) OR (P4)) OR (P3 AND (P5))".

[0083] Since there are no other arcs with Node C as its From_Node, the stack is popped and the current Node is Node D and the current Place is the position after "P2 AND (". The only arc with Node D as its From_Node is Arc 5 with the predicate P5 assigned to it. Since the arc count is one and the To_Node of Arc 5 is the end node. The only operation to be done is inserting P5 at the current place position. The resulting text is "P1 AND ((P2 AND (P5)) OR (P4)) OR (P3 AND (P5))". After Node D is processed, the loop terminates because the stack is empty and the final text value for the Boolean algebra statement has been completed.

[0084] The expansion graph algorithm could be used on its own without the compression algorithm. However, the Boolean text that is created does not always produce a Boolean algebra statement that is the exact mapping of the graph. For example, the same graph where text output from the expansion graph is: ((C1) AND (C3)) OR (C2 AND C3) has the text output from the expansion algorithm after going through the compression algorithm of: (C1 or C2) and C3. The latter Boolean expression is a more efficient representation and more of an exact translation from the corresponding graph.

[0085] The Boolean algebra statement is converted to the graphical flow representation using the Boolean statement to directed graph algorithm in Figs. 19 and 20. The algorithm takes a textural Boolean statement such as (X > 1) AND (Y = 2) and produces a corresponding directed graph. The directed graph can then be displayed or modified by other methods.

[0086] There are two parts to the algorithm. The first part is creating a parse tree of the Boolean expression 230 and the second part is the graph creation algorithm using the parse tree 232. The parse tree consists of leaves and non-leaves. One non-leaf is the root element. All Non-leaves have two children, a left child and a right child. Each element in the parse tree is either a predicate, an OR operator, or an AND operator. The leaves on the tree are always predicates while non-leaves are operators. The creation of the parse tree is known methodology and will not be explained in detail here. After the parse tree is created, the tree is traversed creating the nodes and arcs that make up the directed graph.

[0087] The graph creation algorithm begins by assigning a To_Node and From_Node value to the root of the tree 234. Then the algorithm processes every element in the tree by preordered traversal.

[0088] When the element represents a predicate such as X > 1, then an arc is created connecting the element's From_Node to its To_Node by an arc having the element's predicate assigned to it 236. The From_Node and To_Node were assigned when processing the ele-

ment's parent connector. When the element represents an OR operator, the element's From_Node and To_Node are assigned as From_Node and To_Node for both of the element's children 238. In that way, when the child elements are processed, their To_Node and From_Node values have already been assigned. When the element represents an AND operator, then a new node must be created in between the current From_Node and To_Node. The left-hand child has its From_Node set to the same as the From_Node of this operator. The new node is assigned as the child's To_Node. This same new node is also assigned to be the right child's From_Node. The right child's To_Node is the same as the To_Node of the operator 240. This has the effect of inserting a new node between the elements. In that way, all child elements of AND operators have their To_Node and From_Node values assigned.

[0089] The conversion of a parse tree of a Boolean statement to a directed graph can be further explained with reference to the parse tree 250 in Fig. 21. The parse tree 250 corresponds to the Boolean statement of A<1 or (B<2 and C<3).

[0090] The first step is to initialize the To_Node and From_Node fields of the root element in the tree, "OR". The root nodes' To_Node and From Node are the beginning node and ending node of the graph (Node 1 and Node 2). Next, a preorder traversal is made of the tree.

[0091] A preorder traversal will visit the root element and then the other elements in numeric order as listed in Fig. 21. The first element is the root 251, the second element is root's the left child 252 (since the root's left child has no children), the third element is the root's right child 253, the fourth element is the left child of the root's right child 254, and the fifth element is its right child 255. So, the first element is the root "OR", the second element is A<1, the third is "AND", the fourth is B<2, the fifth is C<3.

[0092] The children of the first element 252 and 253, and the OR operator 251 have their To_Node and From_Node assigned the OR operator's To_Node and From Node values. So, the To_Nodes and From_Nodes of elements two and three, are each assigned Node 1 and Node 2, respectively.

[0093] The next element traversed is the second element 252 which is a predicate with no children. An arc is created with the elements From_Node and To_Node, Node 1 and Node 2, and the resulting arc is assigned the element's predicate, A<1 (260 in Fig. 22).

[0094] The next element, element three 253, is an AND operator. The From_Node of the left child 254 gets this element's From_Node (which is Node 1) and the To_Node of the right child 255 gets this element's To_Node (which is Node 2). A new node is created to be the left child's To_Node and the right child's From_Node. The new node is Node 3.

[0095] The next element, element four 254, is a predicate. An arc is created from this element From_Node to its To_Node (from Node 1 to Node 3) and the resulting arc is assigned the element's predicate, B<2 (262 in Fig. 22). The remaining element, element five 255, is also a predicate. An arc is created from the element's From_Node, Node 2, to its To_Node, Node 3 and the resulting arc is assigned the element's predicate, C<3 (264 in Fig. 22). The resulting graph is the arc list 266 in Fig. 22.

[0096] The graphical flow representation is displayed on the display device using a spacing algorithm. The input for the spacing algorithm is a list of node and predicate arc pairs (produced by the conversion of the Boolean statement to the directed graph). The list of pairs represents a graph of connecting nodes where each connection is described by the predicate text. The algorithm spaces the nodes and corresponding links so that the overlap of paths and the total area covered by the nodes is minimized. This can be illustrated with the following example where there are three nodes. Between nodes 1 and 2 there is a predicate called SALARY>2000 and another predicate called SALARY<500. Between nodes 2 and 3 there is a predicate called JOB=MGR and another predicate called JOB=COOK.

[0097] The algorithm produces from this list two subsequent lists; the first of these is called the node list and the second is the predicate list. The node list contains a list of node objects, each object having a position value. The predicate list contains a list of predicate objects each of which has the position value.

[0098] In general, the algorithm calculates the longest path through the node path, figures out the placement of the nodes on the longest path, the placement of the conditions on the longest path, the placement of remaining nodes on the remaining paths, and placement of remaining conditions on the remaining paths.

[0099] The longest path is calculated by every valid input list containing a description of a chain of nodes linked together via conditions forming in some cases a multitude of paths. Each path starts at the head node (normally 1) and ends at the tail node (normally n), where n is the count of nodes. The longest path is that path or paths originating at the head node and terminating at the tail node, which traverses through the most nodes.

[0100] Using the example and starting with those node pairs which have node 1 as their entry node value, there are two candidates; namely, node 1 to node 2 (SALARY>2000) and node 1 to node 2 (SALARY<500). The first of these nodes can be chosen and the exit node value of 2 is used to form a similar list from the overall predicate pair list giving node 2 to node 3 (JOB=MGR) and node 2 to node 3 (JOB=COOK). The two predicate pairs are supplied which have the value 2 as their entry node value. The first of these is used and the path is followed through to find an exit value equal to 3 which is the tail node. The end of the path is reached going through two predicates giving a path length of 2. This can be used to determine the longest path for a given set of data.

[0101] The next step is the placement of the nodes on the longest path. Given the list of nodes which represent the longest path for a given sample data set, positions are assigned to those nodes found on that longest path. Starting with an arbitrary position by default (100,100) where this number pair will be related to a point on the screen, a special constant is added to the first value for each node in the longest path. This special constant is the length of the longest predicate character string found in the sample data multiplied by the number of points a given character takes on the screen.

[0102] The next step is the placement of the predicates on the longest path. Predicates which have entry and exit nodes which have been assigned position values are now assigned a position themselves. For the simple case where only one predicate exits with a particular entry and exit node pair, the assignment is based on the midpoint between the two points assigned. When a pair of nodes are used by two or more predicates for their entry and exit node values, the assignment requires an extra step which involves adding a constant to the second value attached to the midpoint of a given entry/exit node pair. Further predicates are placed by adding the same constant to the prior value. The assignment procedure is repeated until all predicates situated on the longest path have been positioned.

[0103] The next step is the placement of the remaining nodes not on the longest path. The previous three steps have defined an envelope of space used by the longest path. Any remaining nodes not on the longest path are assigned position values based on the envelop of space. The procedure is straightforward. The entry node's first value is added to the special constant and the envelope's maximum height is used as a second value plus the high constant, giving a position which can be used for any remaining nodes. Each subsequent assignment of a position to a predicate expands the overall envelope of space attached to the sample data. To help minimize the occurrence of cross paths in the generated position of nodes and predicates, a flip component is used in the assignment of all nodes and predicates not found on the longest path. This flip component has the effect of placing each alternate node list on opposite sides of the longest path.

[0104] The final step is the placement of the remaining conditions on the remaining paths. This step makes use of the procedure of the previous step for placement of conditions on the longest path with the variation that the existing space envelope and flip component are taken into account for each placement as well as the nodes involved not being on the longest path.

Claims

1. A method for interactively formulating queries for retrieving data stored in tables having columns and rows in a computerized database system having a central processing unit (CPU), memory, a display device, and a user interaction device, the method comprising the steps of:

receiving input identifying at least one table containing the query data;
receiving input identifying at least one column of the identified table containing the query data;
receiving input identifying at least one row in the identified table containing the query data using a condition statement and characterised in that said condition statement is interactively formulated by the steps of:

(a) displaying a graphical flow representation comprising at least one flowline comprising at least one arc, a predicate assigned to each arc, and a plurality of nodes, each arc positioned between a From_Node node and a To_Node node, wherein predicates on the same flowline are combined through a logical AND operation, and predicates on different flowlines between the same nodes are combined through a logical OR operation;

(b) receiving input specifying a position on a flowline of the graphical flow representation for insertion of a predicate;

(c) receiving input of the predicate;

(d) displaying the graphical flow representation with the predicate inserted in the specified position;

(e) converting the graphical flow representation into a corresponding Boolean algebra statement; and

(f) displaying the Boolean algebra statement.

2. A method as claimed in claim 1 further comprising the steps of:

receiving input from the user adding a predicate to a Boolean algebra statement;
converting the Boolean algebra statement to a corresponding graphical flow representation; and
displaying the graphical flow representation.

3. A method as claimed in claim 1 comprising the further steps for converting a graph that is a graphical flow representation to a Boolean algebra statement comprising the steps of:

(a) compressing the graph to a minimum number of nodes and arcs by combining 2 arcs and replacing said arcs with one single arc, to produce a compressed graph, this combining of 2 arcs being achieved by

combining each pair of arcs having the same To_Nodes and From Nodes into a resulting arc with the predicates assigned to the pair of arcs connected by a logical OR operator assigned as a resulting predicate to the resulting arc; and
combining each pair of arcs having the same To_Node of a first arc and From_Node of a second arc into a resulting arc with the predicates assigned to the pair of arcs connected by an AND operator assigned as a resulting predicate to the resulting arc.; and

(b) creating a corresponding Boolean algebra string by identifying at least one path through the compressed graph from a beginning node to an ending node.

4. A method as claimed in claim 3 wherein the step of creating a corresponding Boolean algebra string further comprises the steps of:

(a) pushing onto a stack a graph path;

(b) popping a path from the stack for further processing;

(c) processing the path to produce a Boolean algebra statement segment while identifying any new paths;

(d) pushing a new path onto the stack; and

(e) repeating steps (b) through (d) until the stack is empty.

5. A method as claimed in claim 2 comprising the further steps for converting a Boolean algebra statement representation to a graphical flow representation of:

(a) arranging the Boolean algebra statement into a parse tree comprising a plurality of elements including a root, each element being a leaf or a non-leaf, each non-leaf having a left child and a right child, and each element having a value assigned for a From_Node and a To_Node, wherein each leaf is assigned a predicate and each non-leaf is assigned an OR operator, or an AND operator; and

(b) processing each element of the tree in preorder traversal assigning each predicate as an arc connecting a From_Node to a To_Node.

6. A method as claimed in claim 5 wherein the processing of each element comprises the steps of:

assigning the root From_Node and To_Node a beginning node and an ending node, respectively, of a resulting graph;
for each OR operator element, assigning the OR operator element From_Node and To_Node as the From_Node and To_Node, respectively, of the left child and right child of the OR operator element;
for each AND operator element:
creating a new node;
assigning the AND operator element From_Node as the From_Node of the left child of the AND operator element;
assigning the new node as the To_Node of the left child and the From_Node of the right child of the AND operator element; and
assigning the AND operator element To_Node as the To_Node of the right child of the AND operator element; and
for each predicate element, creating an arc connecting the From_Node to the To_Node of the predicate element and assigning the predicate to the arc.

7. A method as claimed claim 1 further comprising the step of interactively designating a section of the graphical flow representation as having a logical NOT operation applied to the section.

8. A method for use in a computer system for converting a Boolean algebra statement representation to a graphical representation comprising the steps of:

(a) arranging the Boolean statement into a parse tree wherein each element in the parse tree has a tree From_Node and a tree To_Node and each element is a leaf or a non-leaf, and each non-leaf has a left child and a right child, wherein a leaf is a condition predicate and a non-leaf is an OR operator or an AND operator; and

(b) processing each element of the tree in preorder traversal to assign each condition predicate as a graph arc defined as connecting a graph From_Node to a graph To_Node comprising the steps of:

assigning the root element From_Node and To_Node as the Beginning_Node and Ending_Node of the graph;

for an OR operator element, assigning an OR operator element From_Node and To_Node as the From_Node and To_Node of both of the OR operator element children;

for an AND operator element, creating a New_Node assigning the AND operator element From_Node as the From_Node of the left hand child of the AND operator element, the AND operator element To_Node as the To_Node of the right hand child of the AND operator element, assigning the New_Node as the To_Node of the left hand child and the From_Node of the right hand child of the AND operator element.

9. A method for use in a computer system for converting a graphical representation comprising a plurality of nodes linked by arcs into a Boolean algebra statement representation, comprising the steps of:

(a) when two arcs have the same To_Nodes and From_Nodes, combining the arcs into one arc with the conditions represented by the original arcs linked by a logical OR operand;

(b) when two arcs are in a series with a Third_Node in between having the same two arcs going into and out of it, combining the arcs into one arc with the conditions represented by each original arc linked by an AND operator;

(c) pushing onto a stack a graph path;

(d) popping a path from the stack for further processing;

(e) processing the path to produce a Boolean algebra statement segment while identifying any new paths;

(f) pushing a new path onto the stack; and

(g) repeating steps (d) through (f) until the stack is empty.

10. A computer system comprising: a central processing unit, a memory, a display device, a user interaction device, and means configured to carry out the steps of a method as claimed in any preceding claim.

Patentansprüche

1. Verfahren für die interaktive Erzeugung von Abfragen für den Anruf von Daten, die in Tabellen mit Spalten und Zeilen in einem computergestützten Datenbanksystem gespeichert sind, das eine Zentraleinheit (CPU), einen Speicher, eine Anzeigeeinheit und eine Benutzerinteraktionseinheit beinhaltet, wobei das Verfahren die folgenden Schritte umfasst:

Empfangen einer Eingabe, die mindestens eine Tabelle angibt, die die Abfragedaten enthält;

Empfangen einer Eingabe, die mindestens eine Spalte der angegebenen Tabelle angibt, die die Abfragedaten enthält;

Empfangen einer Eingabe, die mindestens eine Zeile der angegebenen Tabelle, die die Anfragedaten enthält, angibt, unter Verwendung einer Bedingungsanweisung und dadurch gekennzeichnet, dass die Bedingungsanweisung durch die folgenden Schritte interaktiv erzeugt wird:

(a) Anzeigen einer grafischen Datenflussdarstellung, die mindestens eine Flusslinie umfasst, welche mindestens einen Bogen, ein Prädikat, das einem jeden Bogen zugewiesen ist, und eine Vielzahl von Knoten umfasst, wobei jeder Bogen zwischen einem Knoten From_Node und einem Knoten To_Node positioniert ist, wobei Prädikate auf der gleichen Flusslinie durch eine logische UND-Operation verknüpft werden und Prädikate auf verschiedenen Flusslinien zwischen den gleichen Knoten durch eine logische ODER-Operation verknüpft werden.

(b) Empfangen einer Eingabe, die eine Position auf einer Flusslinie der grafischen Datenflussdarstellung für die Einfügung eines Prädikats angibt;

(c) Empfangen einer Eingabe des Prädikats;

(d) Anzeigen der grafischen Datenflussdarstellung mit dem Prädikat, das an der angegebenen Position eingefügt wurde;

(e) Umwandeln der grafischen Datenflussdarstellung in eine entsprechende Boolesche Algebraanweisung; und

(f) Anzeigen der Booleschen Algebraanweisung.

2. Verfahren nach Anspruch 1, das weiter die folgenden Schritte umfasst:

Empfangen einer Eingabe des Benutzers, die einer Booleschen Algebraanweisung ein Prädikat hinzufügt;

Umwandeln der Booleschen Algebraanweisung in eine entsprechende grafische Datenflussdarstellung; und

Anzeigen der grafischen Datenrlussdarstellung.

3. Verfahren nach Anspruch 1, das weiter die Schritte für die Umwandlung eines Graphen, der eine grafische Datenflussdarstellung einer Booleschen Algebraanweisung ist, umfasst und die folgenden Schritte umfasst:

(a) Verdichten des Graphen auf eine Mindestanzahl von Knoten und Bögen, indem zwei Bögen verknüpft werden und die Bögen durch einen einzelnen Bogen ersetzt werden, um einen verdichteten Bogen zu erzeugen, wobei die Verknüpfung von zwei Bögen erzielt wird durch

die Verknüpfung eines Bogenpaars mit den gleichen To_Nodes und From_Nodes zu einem resultierenden Bogen, wobei die Prädikate, die dem Bogenpaar zugewiesen wurden, durch einen logischen ODER-Operator verbunden sind, der als ein resultierendes Prädikat dem resultierenden Bogen zugewiesen wird; und

die Verknüpfung eines jeden Bogenpaars mit dem gleichen To_Node eines ersten Bogens und From_Node eines zweiten Bogens in einen resultierenden Bogen, wobei die Prädikate, die dem Bogenpaar zugewiesen wurden, durch einen UND-Operator verbunden sind, der als ein resultierendes Prädikat dem resultierenden Bogen zugewiesen wird; und

(b) Erzeugen einer entsprechenden Zeichenkette Boolescher Algebra, indem mindestens ein Pfad durch den verdichteten Graphen von einem Startknoten zu einem Endknoten angegeben wird.

4. Verfahren nach Anspruch 3, wobei der Schritt der Erzeugung einer entsprechenden Zeichenkette Boolescher Algebra weiter die folgenden Schritte umfasst:

(a) Auflegen eines Graphenpfads auf einen Stapelspeicher;

(b) Abheben eines Pfads von einem Stapelspeicher für die weitere Verarbeitung;

(c) Verarbeiten des Pfads, um ein Segment einer Booleschen Algebraanweisung zu erzeugen, während etwaige neue Pfade ermittelt werden;

(d) Auflegen eines neuen Pfads auf den Stapelspeicher; und

(e) Wiederholen der Schritte (b) bis (d), bis der Stapelspeicher leer ist.

5. Verfahren nach Anspruch 2, das weiter die folgenden Schritte für die Umwandlung der Darstellung einer Booleschen Algebraanweisung in eine grafische Datenflussdarstellung umfasst:

(a) Anordnen der Booleschen Algebraanweisung in einem Analysebaum, der eine Vielzahl von Elementen einschließlich eines Stamms umfasst, wobei jedes Element entweder ein Blatt oder ein Nichtblatt ist, wobei jedes Nichtblatt ein linkes Kind und ein rechtes Kind hat und wobei jedes Element einen Wert hat, der einem From_Node und einem To_Node zugewiesen ist, wobei jedem Blatt ein Prädikat zugewiesen ist und jedem Nichtblatt ein ODER-Operator oder ein UND-Operator zugewiesen ist; und
(b) Verarbeiten eines jeden Elements des Baums als Durchquerung in festgelegter Reihenfolge, wobei jedes Prädikat als ein Bogen zugewiesen wird, der einen From_Node mit einem To_Node verbindet.

6. Verfahren nach Anspruch 5, wobei die Verarbeitung eines jeden Elements die folgenden Schritte umfasst:

Zuweisen eines Startknotens bzw. einen Endknotens eines resultierenden Graphen zu dem From_Node und To_Node des Stamms;

für jedes ODER-Operatorelement Zuweisen von From_Node und To_Node des ODER-Operatorelements als From_Node bzw. To_Node des linken Kinds und des rechten Kinds des ODER-Operatorelements;

für jedes UND-Operatorelement:

Erzeugen eines neuen Knotens;

Zuweisen des From_Node des UND-Operatorelements als From_Node des linken Kinds des UND-Operatorelements;

Zuweisen des neuen Knotens als To_Node des linken Kinds und als From_Node des rechten Kinds als UND-Operatorelement; und

Zuweisen des To_Node des UND-Operatorelements als To_Node des rechten Kinds des UND-Operatorelements; und

für jedes Prädikatelement Erzeugen eines Bogens, der den From_Node mit dem To_Node des Prädikatelements verbindet, und Zuweisen des Prädikats zu dem Bogen.

7. Verfahren nach Anspruch 1, das weiter den Schritt der interaktiven Kennzeichnung eines Abschnitts der grafischen Datenflussdarstellung als eine logische NICHT-Operation enthaltend, die auf den Abschnitt angewandt wird, umfasst.

8. Verfahren zur Verwendung in einem Computersystem für die Umwandlung einer Darstellung einer Booleschen Algebraanweisung in eine grafische Darstellung, das weiter die folgenden Schritte umfasst:

(a) Anordnen der Booleschen Anweisung in einem Analysebaum, wobei jedes Element des Analysebaums einen From_Node des Baums und einen To_Node des Baums beinhaltet und jedes Element ein Blatt oder ein Nichtblatt ist und jedes Nichtblatt ein linkes Kind und ein rechtes Kind hat, wobei ein Blatt ein Bedingungsprädikat ist und ein Nichtblatt ein ODER-Operator oder ein UND-Operator ist; und

(b) Verarbeiten eines jeden Elements des Baums als Durchquerung in vorgegebener Reihenfolge, um jedes Bedingungsprädikat als einen Graphbogen zuzuweisen, der so definiert ist, dass er den From_Node eines Graphen mit dem To_Node eines Graphen verbindet, das die folgenden Schritte umfasst:

Zuweisen von From_Node und To_Node des Stammelements als Beginning_Node und Ending_Node des Graphen;

für ein ODER-Operatorelement Zuweisen von From_Node und To_Node eines ODER-Operatorelements als From_Node und To_Node beider Kinder des ODER-Operatorelements;

für ein UND-Operatorelement Erzeugen eines New_Node, Zuweisen des From_Node des UND-Operatorelements als From_Node des linken Kinds des UND-Operatorelements, des To_Node des UND-Operatorelements als To_Node des rechten Kinds des UND-Operatorelements, Zuweisen des New_Node als To_Node des linken Kinds und als From_Node des rechten Kinds des UND-Operatorelements.

9. Verfahren zur Verwendung in einem Computersystem für die Umwandlung einer grafischen Darstellung, die eine Vielzahl von Knoten umfasst, die durch Bögen verbunden sind, in eine Darstellung einer Booleschen Algebraanweisung, das die folgenden Schritte umfasst:

(a) Wenn zwei Bögen die gleichen To_Nodes und From_Nodes haben, Verknüpfen der Bögen zu einem Bogen, wobei die Bedingungen, die durch die ursprünglichen Bögen festgelegt sind, durch einen logischen ODER-Operanden verknüpft werden;

(b) Wenn zwei Bögen in einer Reihe mit einem dazwischen liegenden Third_Node, bei dem diese Bögen enden und beginnen, vorhanden sind, Verknüpfen der Bögen zu einem Bogen, wobei die Bedingungen, die durch jeden ursprünglichen Bogen festgelegt sind, durch einen UND-Operator verknüpft werden;

(c) Auflegen eines Graphenpfads auf einen Stapelspeicher;

(d) Abheben eines Pfads von einem Stapelspeicher für die weitere Verarbeitung;

(e) Verarbeiten des Pfads, um ein Segment einer Booleschen Algebraanweisung zu erzeugen, während etwaige neue Pfade ermittelt werden;

(f) Auflegen eines neuen Pfads auf den Stapelspeicher; und

(g) Wiederholen der Schritte (d) bis (f), bis der Stapelspeicher leer ist.

10. Computersystem, das Folgendes umfasst: eine Zentraleinheit, einen Speicher, eine Anzeigeeinheit, eine Benutzerinteraktionseinheit sowie Mittel, die so konfiguriert sind, dass sie die Schritte eines Verfahrens nach einem der vorangehenden Ansprüche ausführen.

Revendications

1. Procédé pour formuler interactivement des requê-

tes relatives à la récupération de données enregistrées dans des tables comportant des colonnes et des rangs dans un système de base de données informatisé comportant une unité centrale, une mémoire, un écran, et un dispositif d'interaction avec l'utilisateur, le procédé comprenant les phases qui consistent à:

recevoir une entrée identifiant au moins une table contenant les données de requête ;

recevoir une entrée identifiant au moins une colonne de la table identifiée comme contenant les données de requête ;

recevoir une entrée identifiant au moins un rang de la table identifiée comme contenant les données de requête à l'aide d'une déclaration de condition et caractérisé en ce que ladite déclaration de condition est formulée interactivement par les phases suivantes :

(a) afficher une représentation graphique de la circulation comprenant au moins une ligne de flux comprenant au moins un arc, un prédicat attribué à chaque arc et une pluralité de noeuds, chaque arc étant positionné entre un noeud From_Node (expéditeur) et un noeud To_Node (destinataire), où les prédicats présents sut une même ligne sont combinés par une opération logique ET, et les prédicats situés sur différentes lignes de circulation entre les mêmes noeuds sont combinés au moyen d'une opération logique OU ;

(b) recevoir une entrée spécifiant une position sur une ligne de flux de la représentation graphique de la circulation pour l'insertion d'un prédicat ;

(c) recevoir l'entrée du prédicat ;

(d) afficher la représentation graphique de la circulation avec le prédicat inséré à la position spécifiée ;

(e) convertir la représentation graphique de la circulation en une instruction d'algèbre booléenne correspondante ; et

(f) visualiser l'énoncé d'algèbre booléenne.

2. Procédé selon la revendication 1 comprenant en outre les phases qui consistent à :

recevoir une entrée de l'utilisateur ajoutant un prédicat à un énoncé d'algèbre booléenne ;

convertir l'énoncé d'algèbre booléenne en une représentation graphique de circulation correspondante ; et

afficher à l'écran la représentation graphique de la circulation.

3. Procédé selon la revendication 1 comprenant en outre les phases pour convertir en un énoncé d'algèbre booléenne un graphique qui est une représentation graphique de circulation, comprenant les phases qui consistent à :

(a) comprimer le graphique en un nombre minimal de noeuds et d'arcs en combinant 2 arcs et en remplaçant les dits arcs par un seul, pour produire un graphique comprimé, cette combinaison de 2 arcs étant réalisée en combinant chaque paire d'arcs ayant les mêmes To_Nodes (noeuds destinataires) et From_Nodes (noeuds expéditeurs) en un arc résultant, les prédicats attribués à la paire d'arcs et connectés par un opérateur logique OU étant attribués en tant que prédicat résultant à l'arc résultant ; et en combinant chaque paire d'arcs ayant les mêmes To_Node (noeud destinataire) qu'un premier arc et From_Node (noeud expéditeur) qu'un deuxième arc en un arc résultant, les prédicats attribués à la paire d'arcs, connectés par un opérateur ET, étant attribués à l'arc résultant comme prédicat résultant ; et

(b) créer une chaîne algébrique booléenne correspondante en identifiant au moins un chemin dans le graphique comprimé allant d'un noeud de départ à un noeud de fin.

4. Procédé selon la revendication 3 où la phase de création d'une chaîne algébrique booléenne correspondante comprend en outre les phases suivantes :

(a) empiler un chemin graphique sur une pile ;

(b) extraire un chemin de la pile en vue d'un traitement ultérieur ;

(c) traiter le chemin pour produire un segment d'énoncé d'algèbre booléenne tout en identifiant tout nouveau chemin ;

(d) empiler un nouveau chemin sur la pile ; et

(e) répéter les phases (b) à (d) jusqu'à ce que la pile soit vide.

5. Procédé selon la revendication 2 comprenant en

outre les phases suivantes pour convertir la représentation d'un énoncé d'algèbre booléenne en une représentation graphique de circulation :

(a) arranger l'énoncé d'algèbre booléenne en un arbre d'analyse comprenant une pluralité d'éléments y compris une racine, chaque élément étant une feuille ou une non-feuille, chaque non-feuille ayant un enfant gauche et un enfant droit, et chaque élément ayant une valeur attribuée pour un From_Node (noeud expéditeur) et un To_Node (noeud destinataire), où à chaque feuille est attribué un prédicat et à chaque non-feuille un opérateur OU ou un opérateur ET ; et

(b) traiter chaque élément de l'arbre en traversée préordonnée en assignant chaque prédicat en tant qu'arc connectant un From_Node (noeud expéditeur) à un To_Node (noeud destinataire).

6. Procédé selon la revendication 5, où le traitement de chaque élément comprend les phases qui consistent à :

attribuer respectivement au From_Node racine (noeud expéditeur) et au To_Node racine (noeud destinataire) un noeud de départ et un noeud d'arrivée d'un graphique résultant ;
pour chaque opérateur OU, assigner le From_Node (noeud expéditeur) et le To_Node (noeud destinataire) de l'opérateur OU en tant que, respectivement, From_Node (noeud expéditeur) et To_Node (noeud destinataire) de l'enfant gauche et de l'enfant droit de l'opérateur OU ;
pour chaque opérateur ET :
créer un nouveau noeud ;
assigner le From_Node (noeud expéditeur) de l'opérateur ET en tant que From_Node (noeud expéditeur) de l'enfant gauche de l'opérateur ET ;
assigner le nouveau noeud en tant que To_Node (noeud destinataire) de l'enfant gauche et From_Node (noeud expéditeur) de l'enfant droit de l'opérateur ET ; et
assigner le To_Node (noeud destinataire) de l'opérateur ET en tant que To_Node (noeud destinataire) de l'enfant droit de l'opérateur ET ; et
pour chaque prédicat, créer un arc connectant le From_Node (noeud expéditeur) au To_Node (noeud destinataire) du prédicat et assigner le prédicat à l'arc.

7. Procédé selon la revendication 1 comprenant en outre la phase qui consiste à désigner interactivement une partie de la représentation graphique de la circulation comme étant une partie sur laquelle est appliquée une opération logique NON.

8. Procédé à utiliser dans un système informatique, pour convertir la représentation d'un énoncé d'algèbre booléenne en une représentation graphique comprenant les phases suivantes :

(a) arranger l'énoncé booléen en un arbre d'analyse dans lequel chaque élément de l'arbre d'analyse a un From_Node (noeud expéditeur) et un To_Node (noeud destinataire) dans l'arbre et où chaque élément est une feuille ou une non-feuille, et chaque non-feuille a un enfant gauche et un enfant droit, où une feuille est un prédicat de condition et une non-feuille est un opérateur OU ou un opérateur ET ; et

(b) traiter chaque élément de l'arbre en traversée préordonnée pour attribuer chaque prédicat de condition en tant qu'arc graphique connectant un From_Node (noeud expéditeur) graphique à un To_Node (noeud destinataire) graphique, au moyen des phases suivantes :

assigner le From_Node (noeud expéditeur) racine et le To_Node (noeud destinataire) racine en tant que noeud de départ (Beginning_Node) et noeud d'arrivée (Ending_Node) du graphique ;

pour un opérateur OU, assigner le From_Node (noeud expéditeur) et le To_Node (noeud destinataire) d'un opérateur OU en tant que From_Node (noeud expéditeur) et To_Node (noeud destinataire) des deux enfants de l'élément-opérateur OU ;

pour un opérateur ET, créer un nouveau noeud (New_Node), assigner le From_Node (noeud expéditeur) de l'opérateur ET comme From_Node (noeud expéditeur) de l'enfant gauche de l'élément-opérateur ET, le To_Node (noeud destinataire) de l'opérateur ET en tant que To_Node (noeud destinataire) de l'enfant droit de l'opérateur ET, assigner le New_Node (nouveau noeud) en tant que To_Node (noeud destinataire) de l'enfant gauche et en tant que From_Node (noeud expéditeur) de l'enfant droit de l'opérateur ET.

9. Procédé à utiliser dans un systeme informatique pour convertir une représentation graphique com-

prenant une pluralité de noeuds reliés par des arcs en une représentation d'énoncé d'algèbre booléenne, comprenant les phases qui consistent à :

(a) quand deux arcs ont les mêmes To_Nodes (noeuds destinataires) et From_Nodes (noeuds expéditeurs), combiner les arcs en un arc, les conditions représentés par les arcs d'origine étant liées par un opérateur logique OU ;

(b) quand deux arcs sont en série avec un Third_Node (troisième noeud) présent entre eux avec les deux mêmes arcs entrant et sortant de lui, combiner les arcs en un arc, les conditions représentées par chaque arc originel étant reliées par un opérateur ET ;

(c) empiler un chemin graphique sur une pile ;

(d) extraire un chemin de la pile ne vue de son traitement ;

(e) traiter le chemin pour produire un segment d'enoncé d'algèbre booléenne tout en identifiant tout nouveau chemin ;

(f) empiler un nouveau chemin sur la pile ; et

(g) répéter les phases (d) à (f) jusqu'à ce que la pile soit vide.

10. Système informatique comprenant : comportant une unité centrale, une mémoire, un écran, et un dispositif d'interaction avec l'utilisateur, et un moyen configuré pour exécuter les phases d'un procédé tel que revendiqué dans l'une quelconque des revendications précédentes.

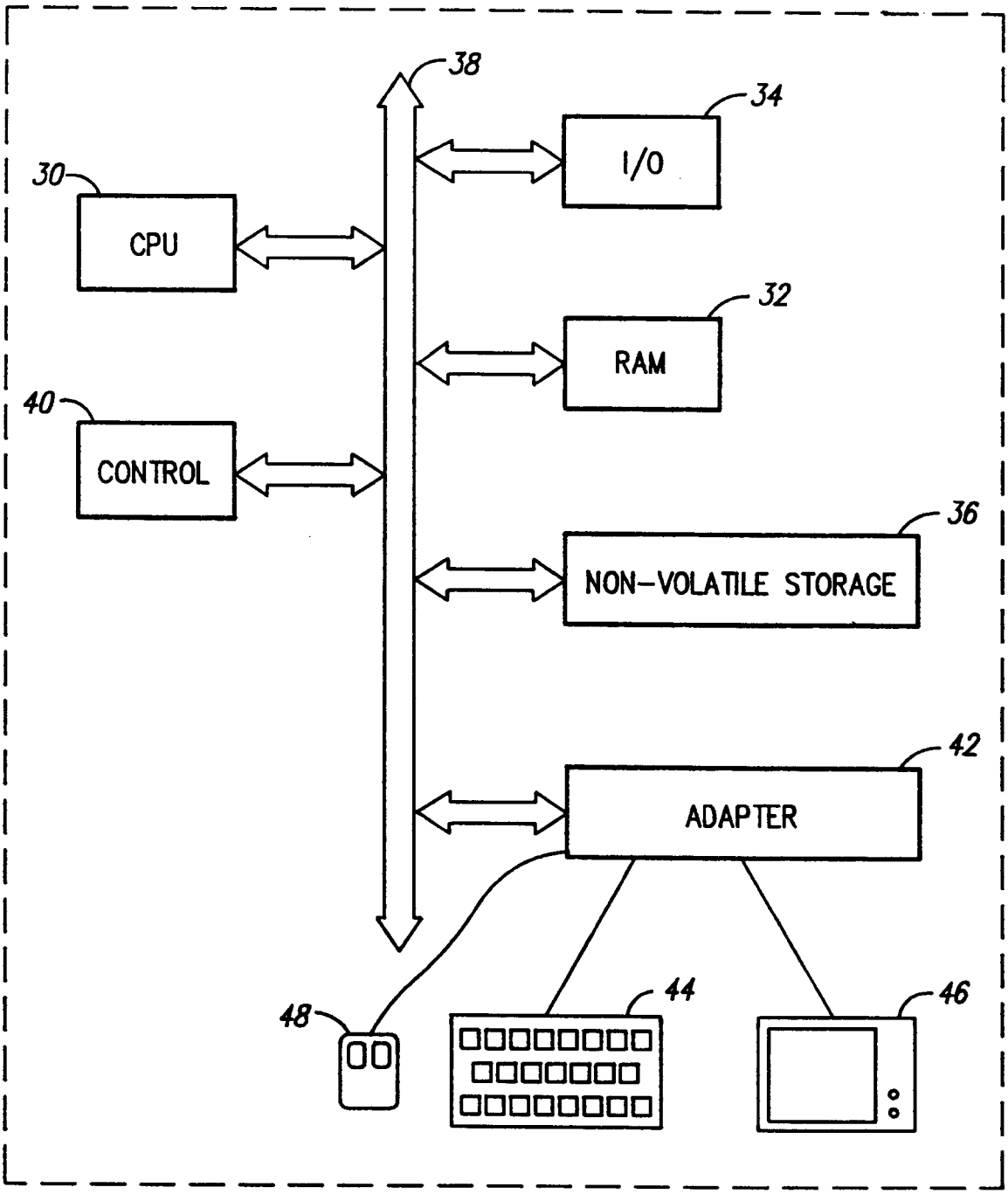

38
34
I/O
30
CPU
32
RAM
40
CONTROL
36
NON-VOLATILE STORAGE
42
ADAPTER
48
44
46

FIG. 1

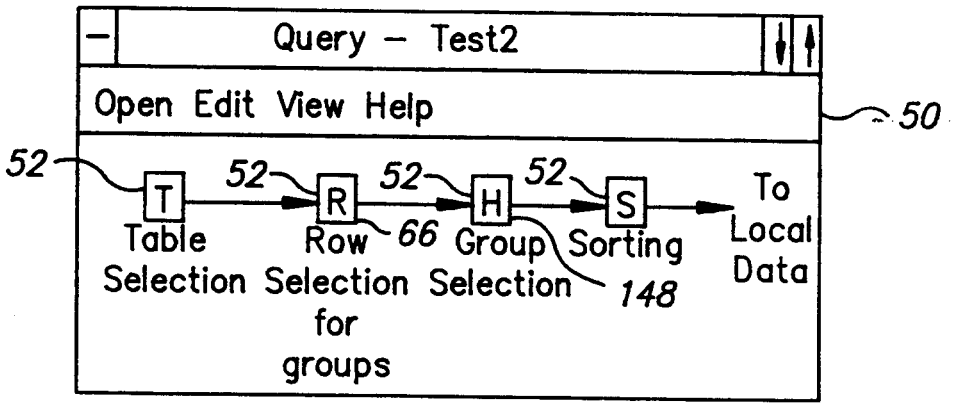

Query – Test2
Open Edit View Help
50
52
T
Table Selection
52
R
Row Selection for groups
66
52
H
Group Selection
148
52
S
Sorting
To Local Data

FIG. 2

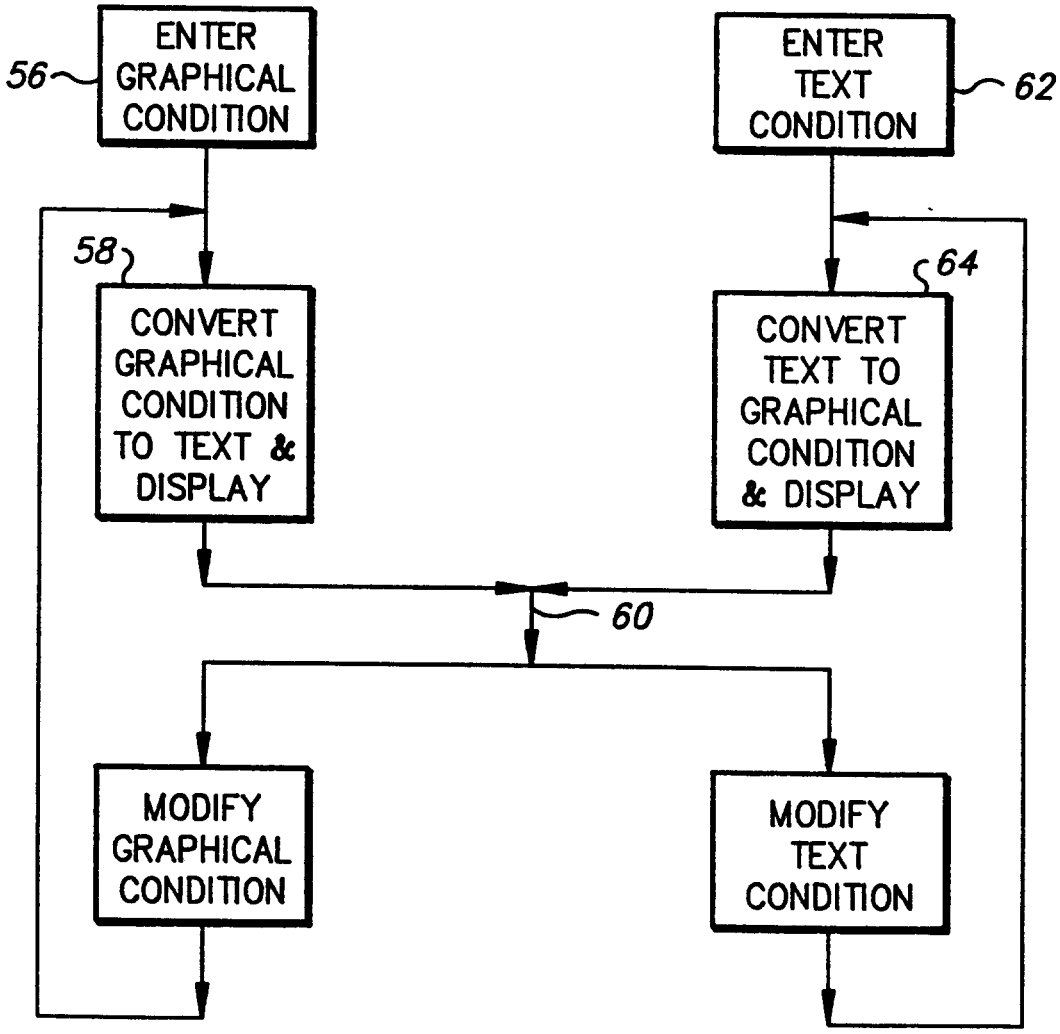

56
ENTER GRAPHICAL CONDITION
58
CONVERT GRAPHICAL CONDITION TO TEXT & DISPLAY
ENTER TEXT CONDITION
62
64
CONVERT TEXT TO GRAPHICAL CONDITION & DISPLAY
60
MODIFY GRAPHICAL CONDITION
MODIFY TEXT CONDITION

FIG. 3

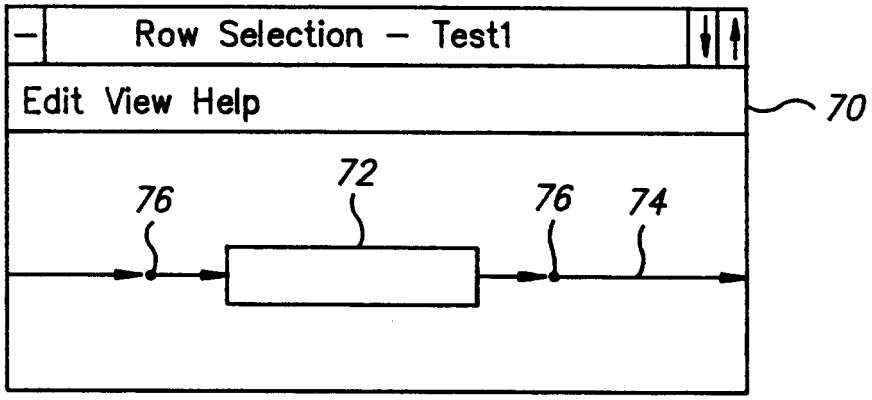

Row Selection – Test1
Edit View Help
70
72
76
76
74

FIG. 4

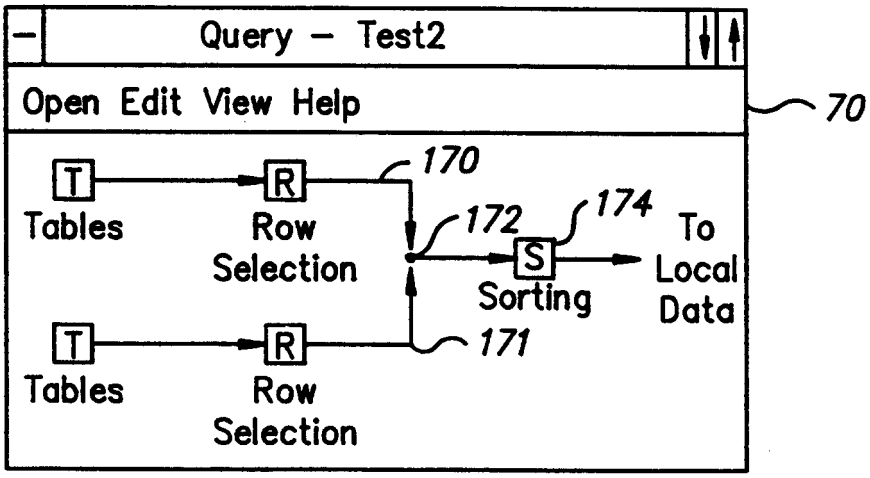

Query – Test2
Open Edit View Help
70
T
Tables
R
Row Selection
170
172
174
S
Sorting
To Local Data
T
Tables
R
Row Selection
171

FIG. 11

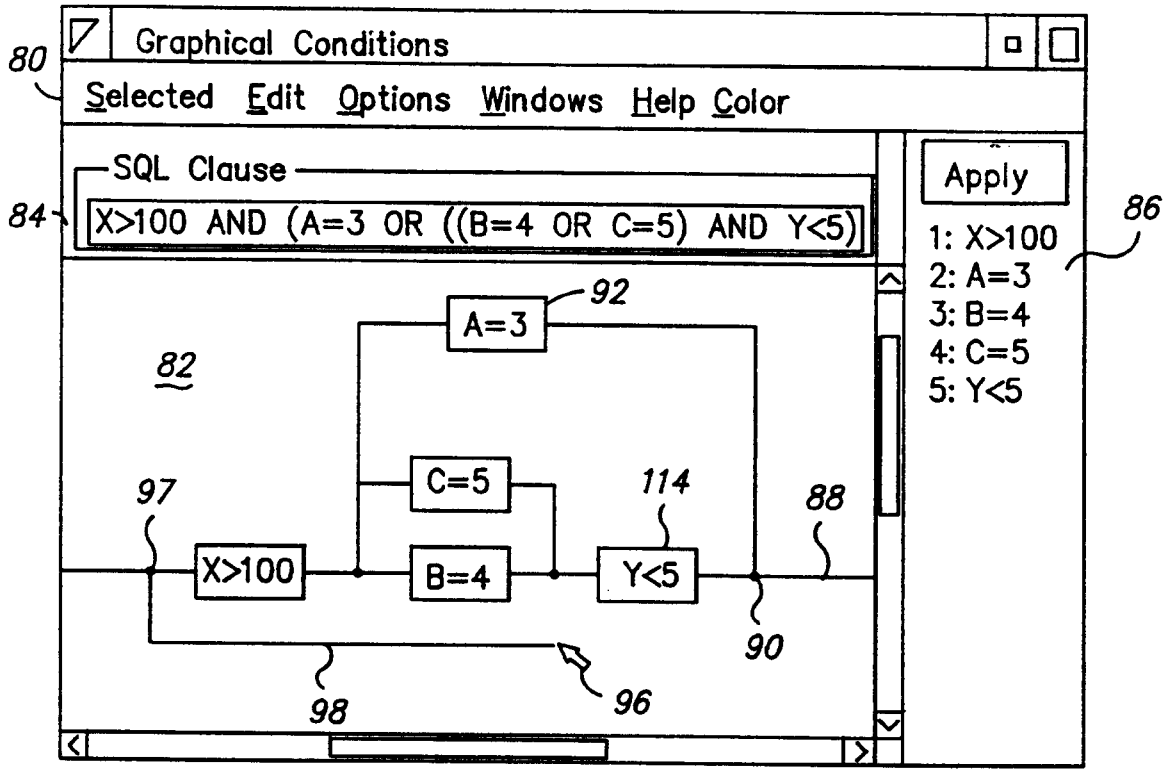

Graphical Conditions
Selected Edit Options Windows Help Color
SQL Clause
X>100 AND (A=3 OR ((B=4 OR C=5) AND Y<5)
Apply
1: X>100
2: A=3
3: B=4
4: C=5
5: Y<5
80
82
84
86
88
90
92
96
97
98
114
A=3
C=5
B=4
X>100
Y<5

FIG. 5

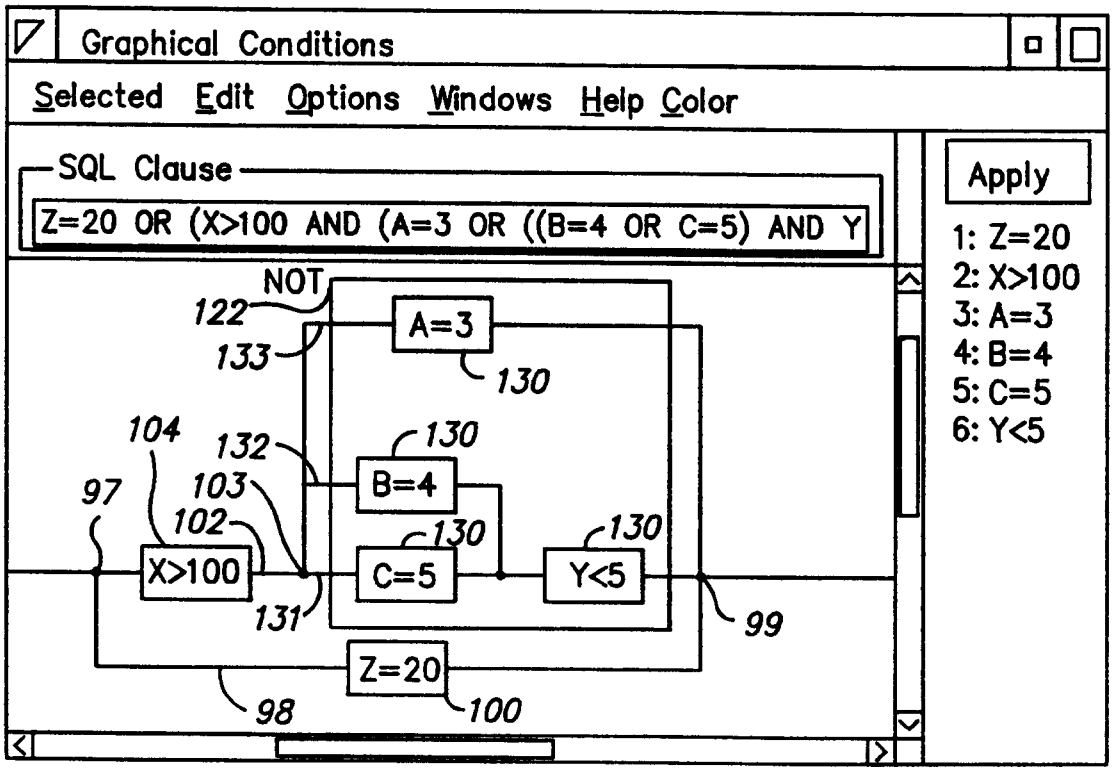

Graphical Conditions
Selected Edit Options Windows Help Color
SQL Clause
Z=20 OR (X>100 AND (A=3 OR ((B=4 OR C=5) AND Y
Apply
1: Z=20
2: X>100
3: A=3
4: B=4
5: C=5
6: Y<5
NOT
122
133
130
104
132
103
102
131
97
98
99
100
A=3
B=4
C=5
X>100
Y<5
Z=20

FIG. 6

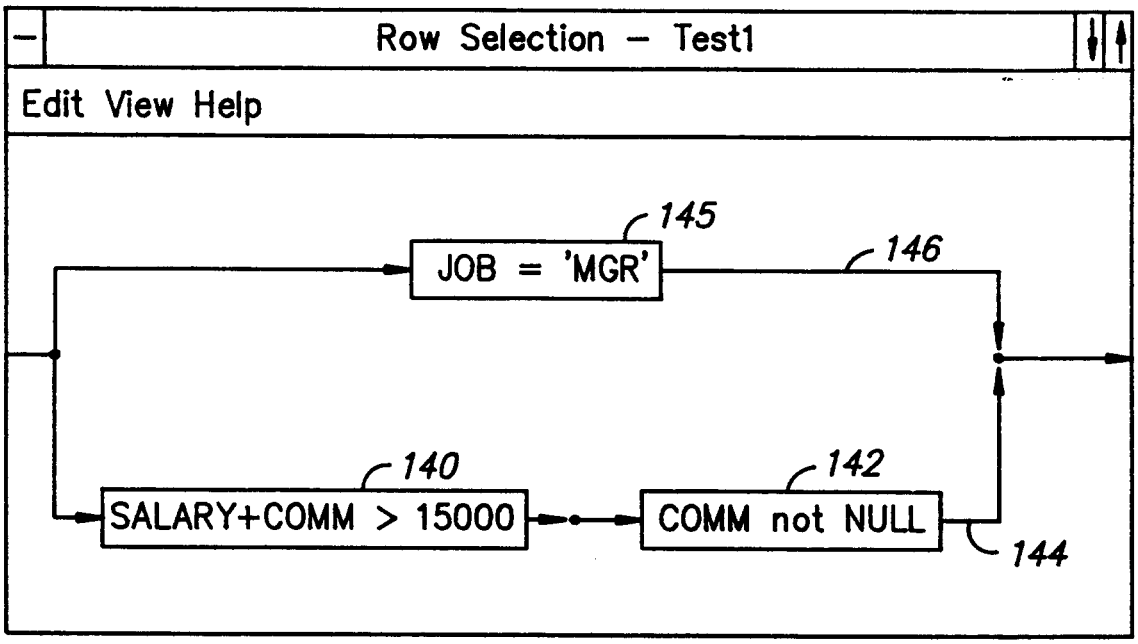

Row Selection – Test1
Edit View Help
145
JOB = 'MGR'
146
140
SALARY+COMM > 15000
142
COMM not NULL
144

FIG. 7

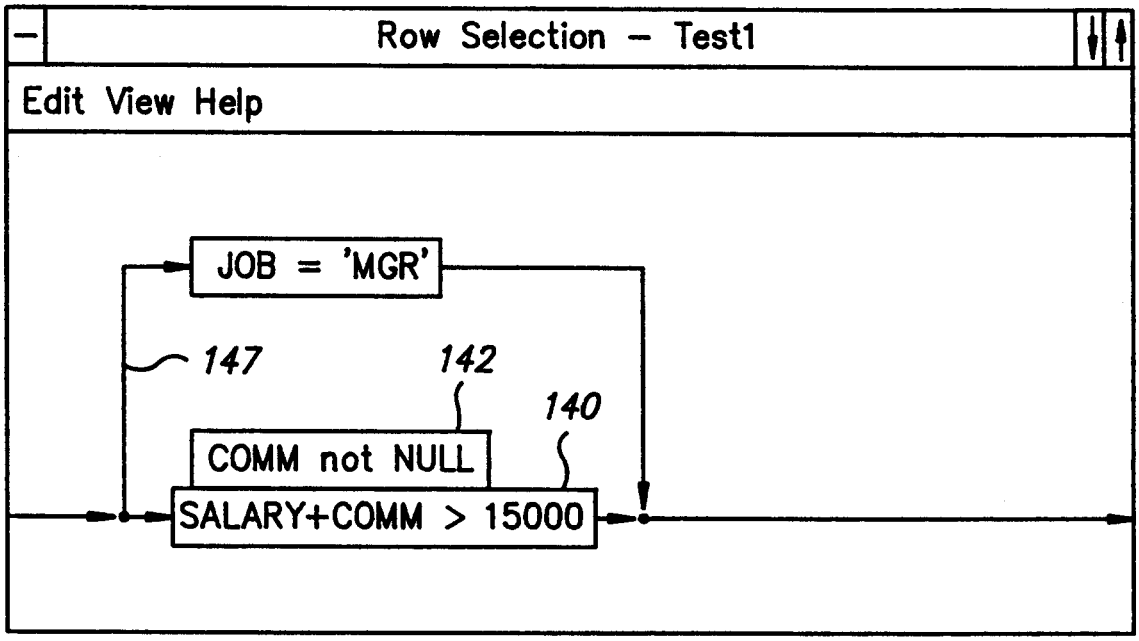

Row Selection – Test1
Edit View Help
JOB = 'MGR'
147
142
140
COMM not NULL
SALARY+COMM > 15000

FIG. 8

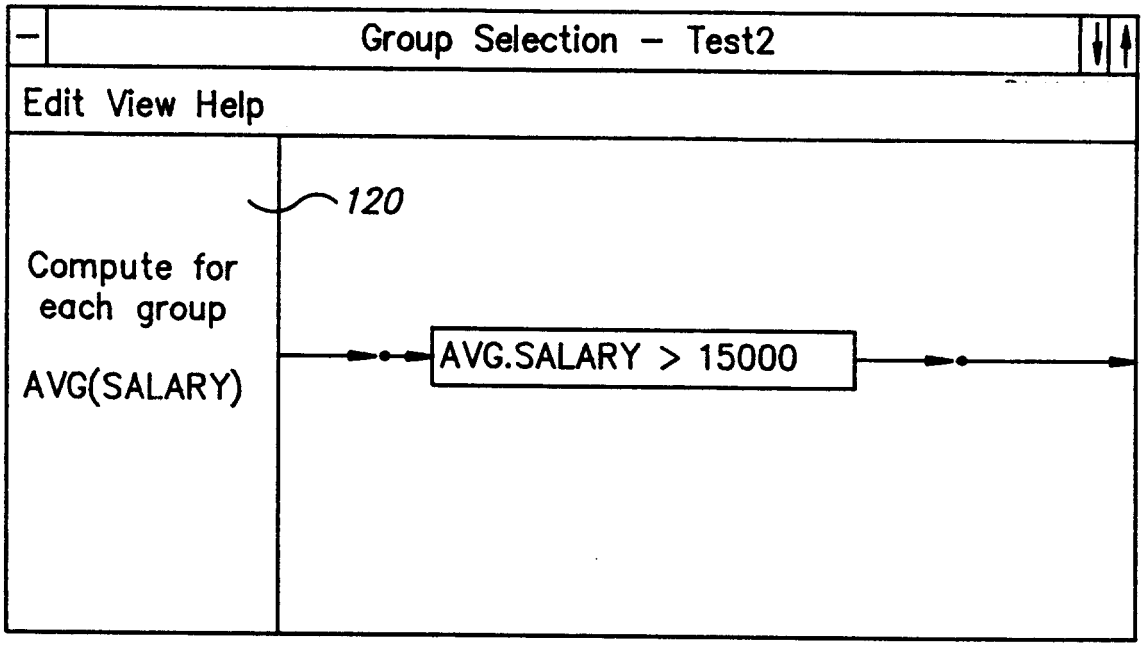

Group Selection – Test2
Edit View Help
120
Compute for each group
AVG(SALARY)
AVG.SALARY > 15000

FIG. 9

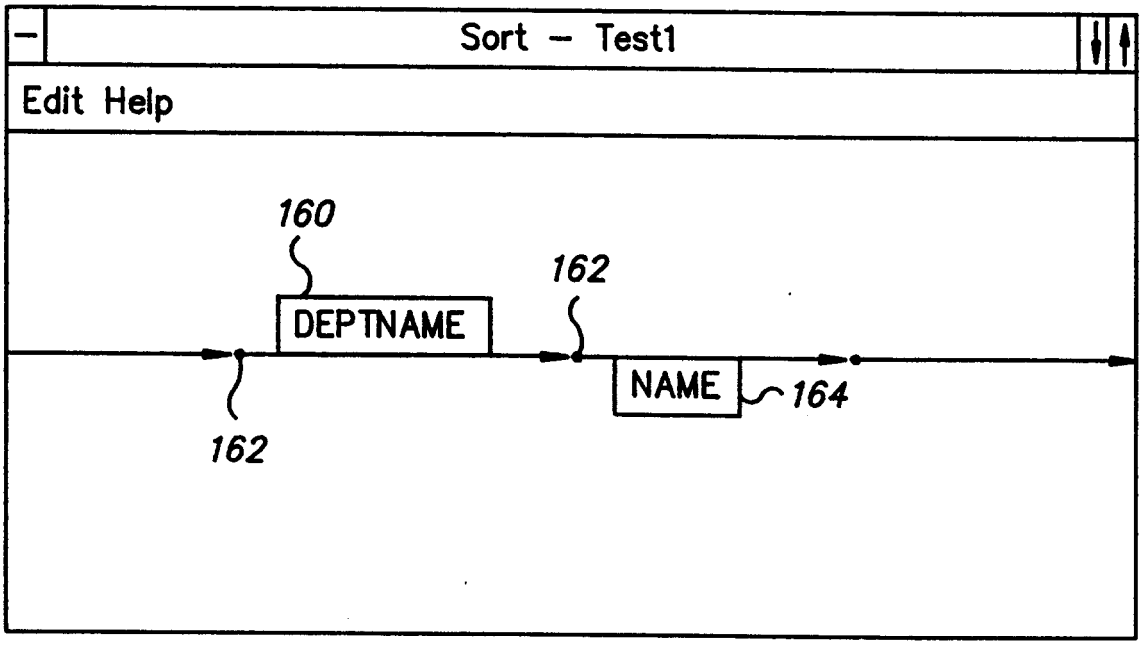

Sort – Test1
Edit Help
160
162
DEPTNAME
NAME
164
162

FIG. 10

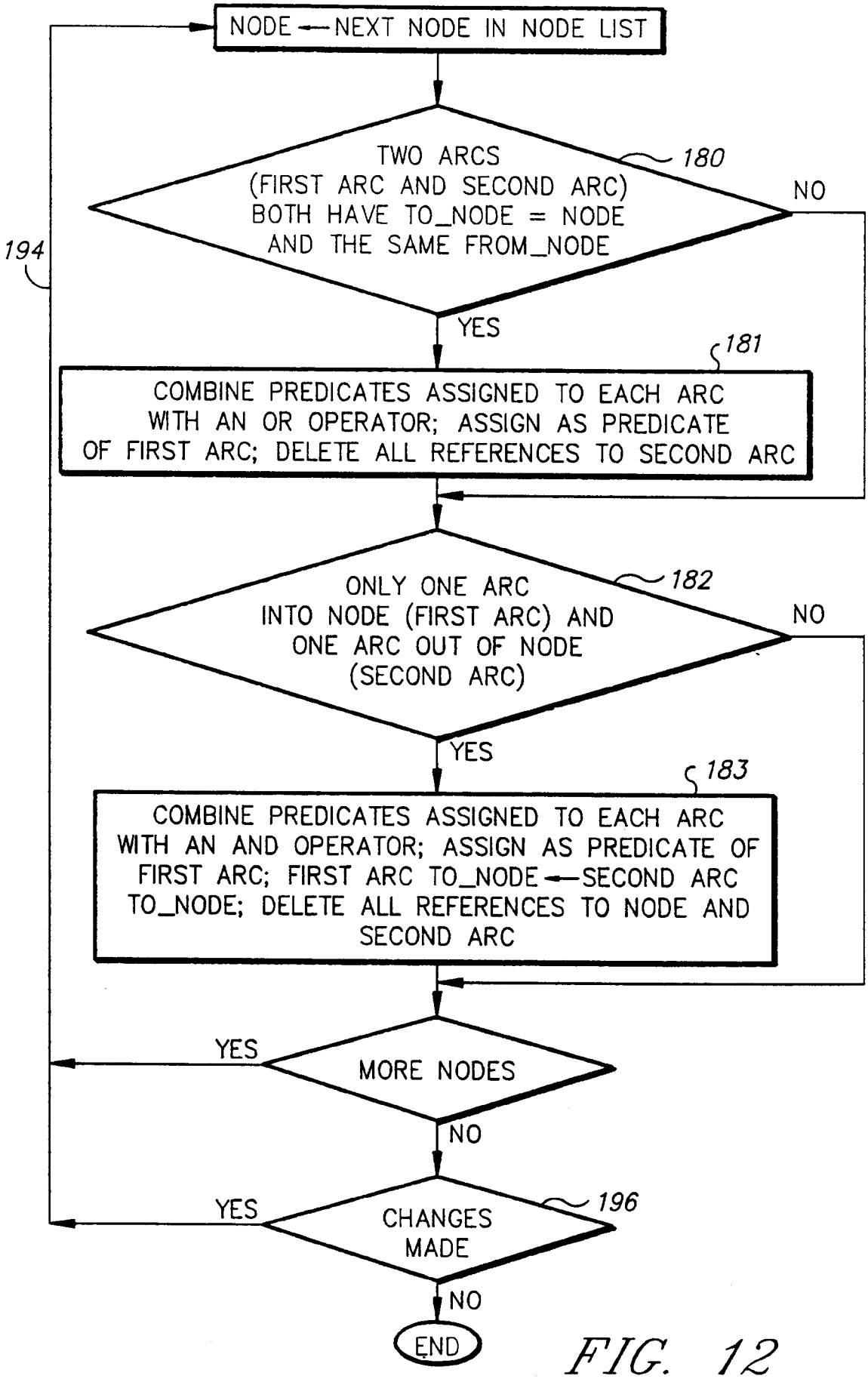
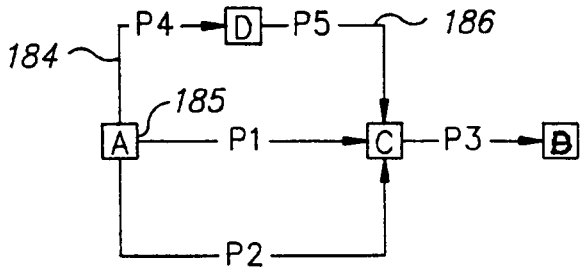

NODE ← NEXT NODE IN NODE LIST
TWO ARCS (FIRST ARC AND SECOND ARC) BOTH HAVE TO_NODE = NODE AND THE SAME FROM_NODE
180
NO
YES
194
181
COMBINE PREDICATES ASSIGNED TO EACH ARC WITH AN OR OPERATOR; ASSIGN AS PREDICATE OF FIRST ARC; DELETE ALL REFERENCES TO SECOND ARC
ONLY ONE ARC INTO NODE (FIRST ARC) AND ONE ARC OUT OF NODE (SECOND ARC)
182
NO
YES
183
COMBINE PREDICATES ASSIGNED TO EACH ARC WITH AN AND OPERATOR; ASSIGN AS PREDICATE OF FIRST ARC; FIRST ARC TO_NODE ← SECOND ARC TO_NODE; DELETE ALL REFERENCES TO NODE AND SECOND ARC
YES
MORE NODES
NO
YES
CHANGES MADE
196
NO
END

FIG. 12

```
do until No_Changes = true
 No_Changes = true

 do for each node in Node Array
  if In_node > 1 then ~ 197                                  180
    if two arcs in In_List_node have same from node
      Combine condition text between two arcs with '(text1)OR(text2)'
181~ Place combined text into first arc
      Delete all references to second arc
      No_Changes = false
    endif
  else                                                  182
    in In_node = 1 AND Out_node = 1 then
      Combine condition text between two arcs with '(text1)AND(text2)'
183~ Place combined text into first arc
      Delete all references to second arc
      No_Changes = false
    endif
  endif
 enddo
enddo
```

FIG. 13



187 — 188 189 190 191

Nodes					Arcs			
Node	In	Out	In List	Out List	#	From Node	To Node	Condition
A	0	3	–	1,2,4	1	A	C	C1
B	1	0	3	–	2	A	C	C2
C	3	1	1,2,5	3	3	C	B	C3
D	1	1	4	5	4	A	D	C4
					5	D	C	C5

FIG. 14

A –(((C1)or(C2))or((c4)and(c5))) and (c3))→ B — 200

Nodes					Arcs			
Node	In	Out	In List	Out List	#	From Node	To Node	Condition
A	0	1	–	1	1	A	B	(((C1)or(C2))OR((C4) and(C5)))and(C3)
B	1	0	1	–				
C	0	0	–	–				
D	0	0	–	–				

FIG. 15

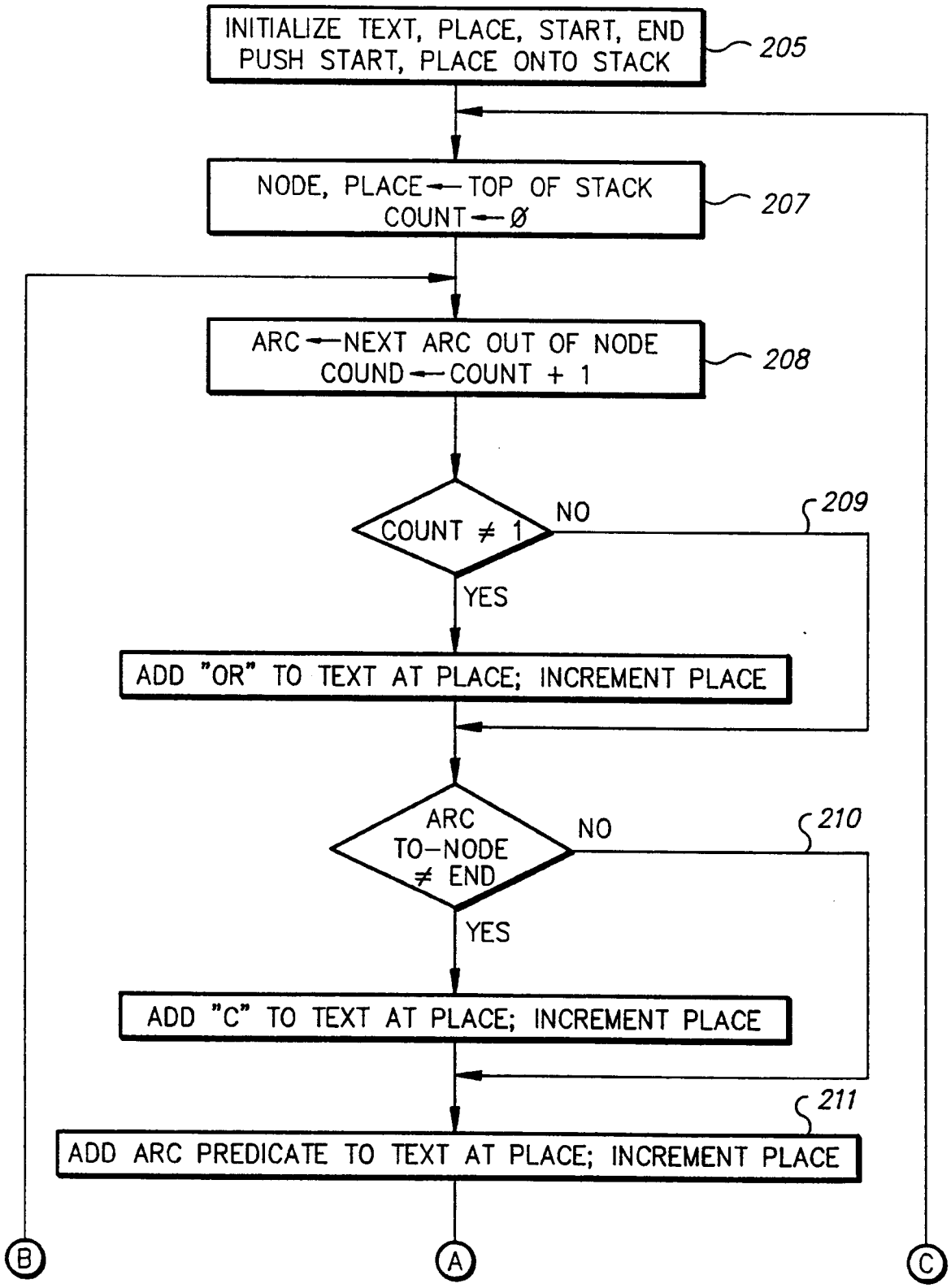

INITIALIZE TEXT, PLACE, START, END
PUSH START, PLACE ONTO STACK
205
NODE, PLACE ← TOP OF STACK
COUNT ← Ø
207
ARC ← NEXT ARC OUT OF NODE
COUND ← COUNT + 1
208
COUNT ≠ 1
NO
209
YES
ADD "OR" TO TEXT AT PLACE; INCREMENT PLACE
ARC TO-NODE ≠ END
NO
210
YES
ADD "C" TO TEXT AT PLACE; INCREMENT PLACE
211
ADD ARC PREDICATE TO TEXT AT PLACE; INCREMENT PLACE
B
A
C

FIG. 16A

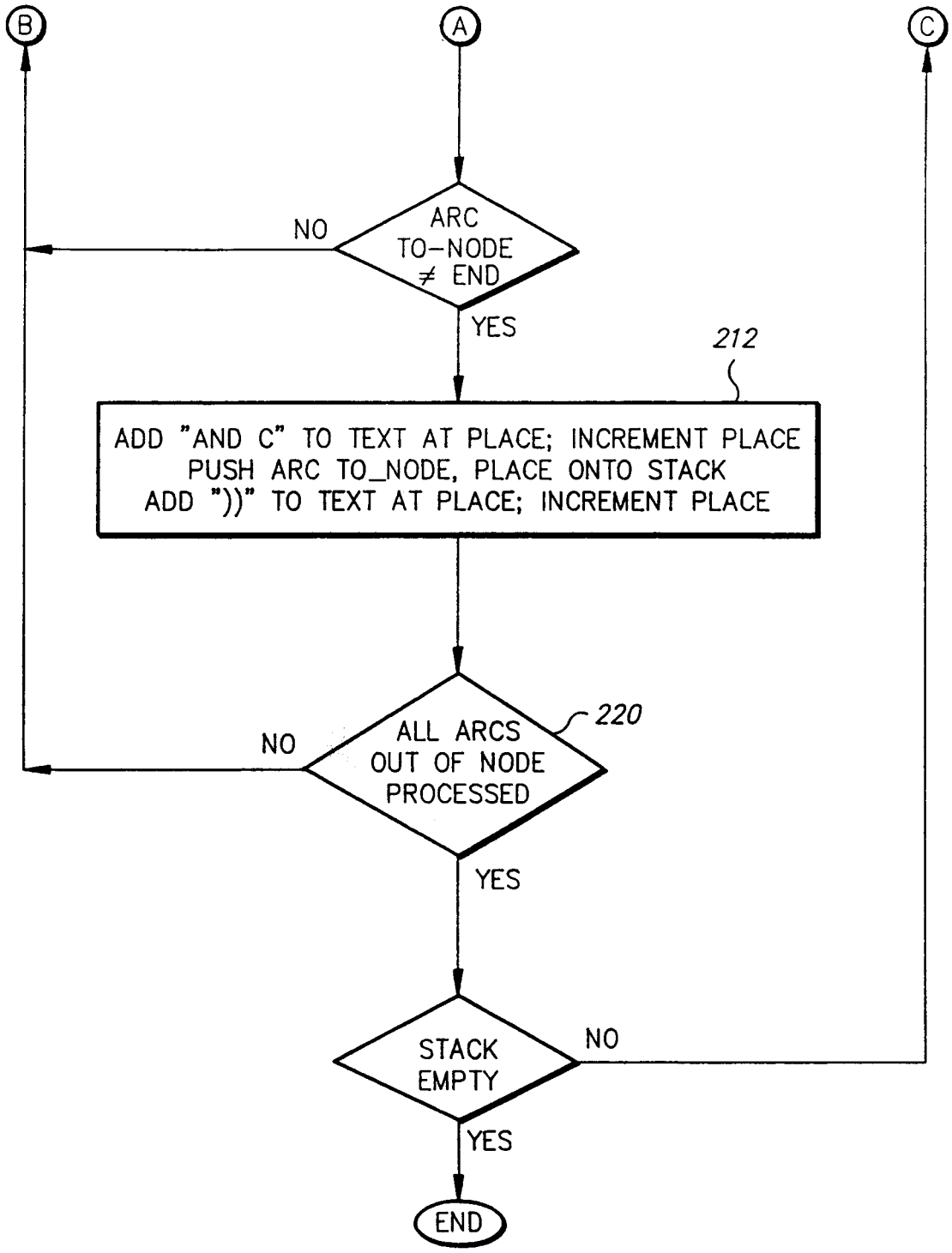
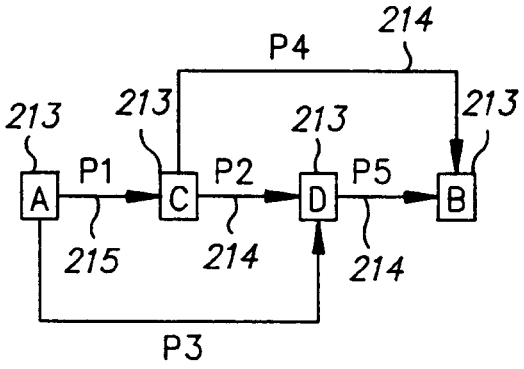

B
A
C
ARC
TO-NODE
≠ END
NO
YES
212
ADD "AND C" TO TEXT AT PLACE; INCREMENT PLACE
PUSH ARC TO_NODE, PLACE ONTO STACK
ADD "))" TO TEXT AT PLACE; INCREMENT PLACE
220
ALL ARCS
OUT OF NODE
PROCESSED
NO
YES
STACK
EMPTY
NO
YES
END

FIG. 16B

```
Start_Node = starting node of graph (IN=0 and OUT>0)
End_Node = ending node of graph (IN>0 and OUT=0)
Boolean_Text =
Current_Place = 1
push Start_Node and Current_Place onto stack ~ 205

do while (stack not empty)
 Current_Node = pop top node off of stack
 Current_Place = pop top place off of stack ~ 207
 Arc_Count = 0

do for each are
 if From_Node_Arc = Current_Node then
  Arc_Count = Arc_Count +1

  If Arc_Count -= 1 then
   Current_Place = add_text Current_Place, ()R' ) ~ 209
  endif
  if 'To_Node'_Arc -= End_Node then
   Current_Place = add_text( Current_Place, '(') ~ 210

  Current_Place = add_text( Current_Place.Condition_Arc) ~ 211
  if 'To_Node'_Arc -= End_Node then
   Current_Place = add_text( Current_Place, 'AND')
   Current_Place = add_text( Current_Place, '(')

   Push 'To-Node'_Arc and Current_Place onto stack ~ 212
   Current_Place = add_text( Current_Place. ')')
   Current_Place = add_text( Current_Place. ')')
  endif
 endif
enddo
enddo


procedure add_text( Location, Text)
 /*add_text inserts the string 'Text' into Boolean_Text at Location,*/
 /*returning a new location just AFTER the inserted text.          */

 Boolean_text = INSERT(Boolean_text, Text, Location)
 New_Location = Location just AFTER the inserted text

 return New_Location
```

FIG. 17



215 214 216

Nodes	Arcs				Stack
213 ~ Node	#	From Node	To Node	Condition	Node.Place
A	1	A	C	P1	A.1
B	2	C	D	P2	
C	3	A	D	P3	
D	4	C	B	P4	
	5	D	B	P5	

Boolean_Text = ' '
Current_Node = A
Current_Place = 1
Start_Node = A
End_Node = B

FIG. 18

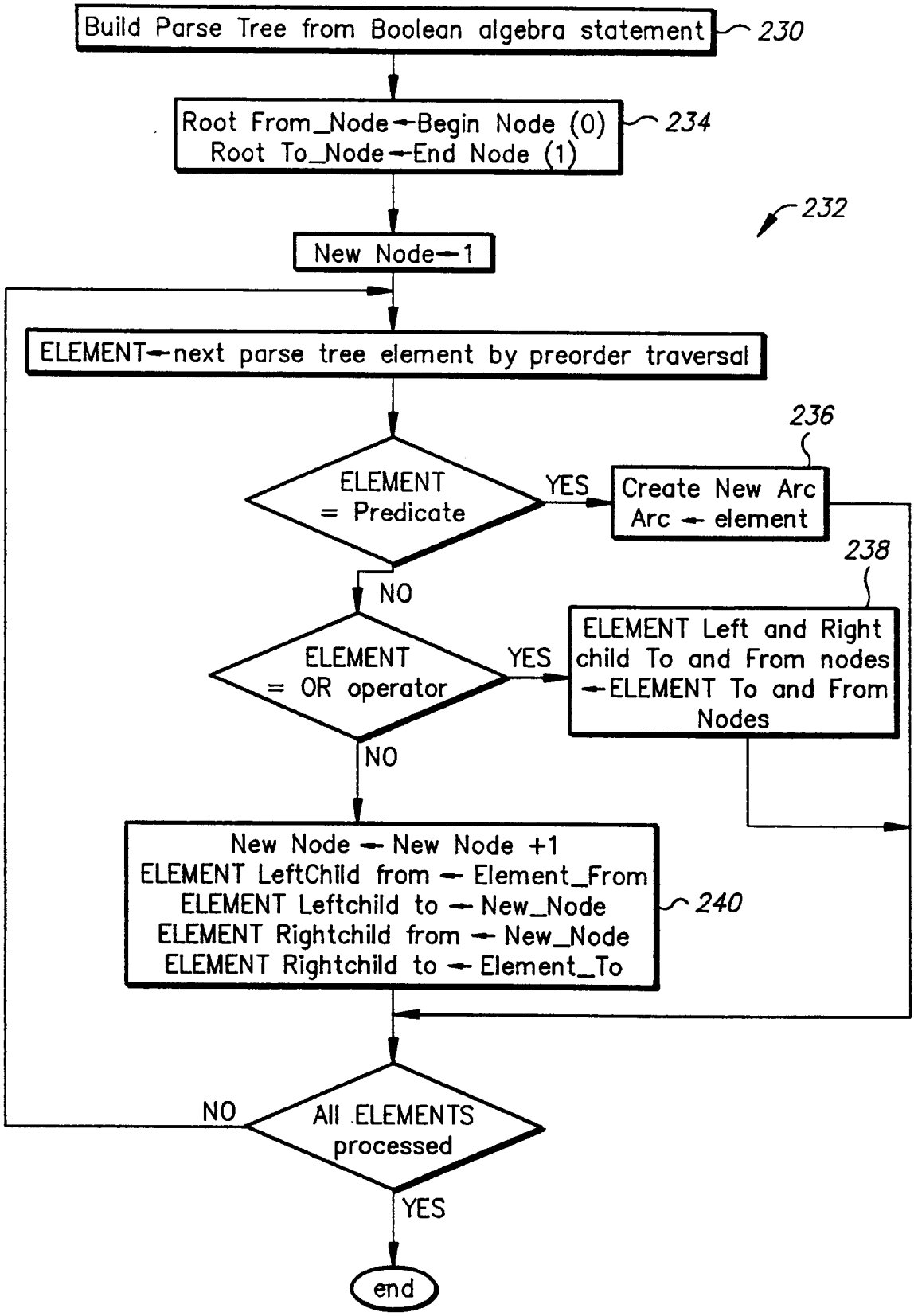
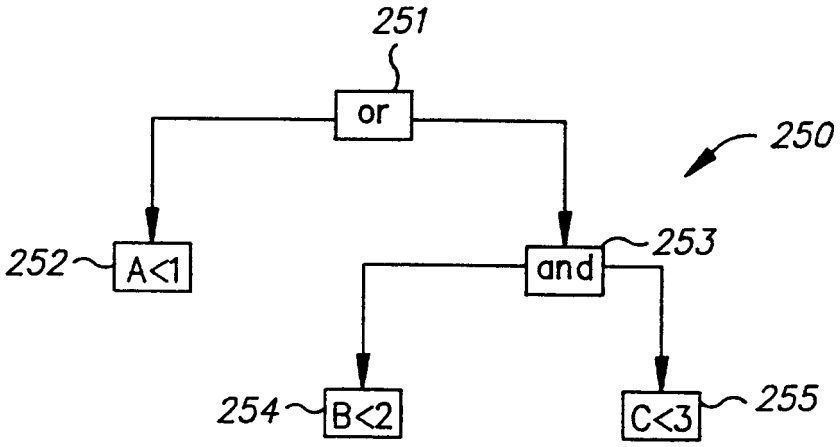

Build Parse Tree from Boolean algebra statement
230
Root From_Node←Begin Node (0)
Root To_Node←End Node (1)
234
232
New Node←1
ELEMENT←next parse tree element by preorder traversal
236
ELEMENT = Predicate
YES
Create New Arc
Arc ← element
238
NO
ELEMENT = OR operator
YES
ELEMENT Left and Right child To and From nodes ←ELEMENT To and From Nodes
NO
New Node ← New Node +1
ELEMENT LeftChild from ← Element_From
ELEMENT Leftchild to ← New_Node
ELEMENT Rightchild from ← New_Node
ELEMENT Rightchild to ← Element_To
240
NO
All ELEMENTS processed
YES
end

FIG. 19

```
Build parse tree for Boolean condition                    ~ 230

Next_Node_Id - 0
Parse_Tree_Node = Root of parse tree

* Set up first node of tree for processing                ~ 234
Parse_TreeNode from_node = Next_Node_ID
Next_Node = Next_Node_Id + 1
Parse_Tree_Node to_node = Next_Node_Id

DO for each Parse_Tree_Node BY preorder traversal

  /* All simple conditions (X>1) have no children                */
  If Parse_Tree_Node has no children THEN
    Create graph arc where
      "To Node" is Parse_Node to_node                       ~ 6
      "From Node" is Parse_Tree_Node from_node
      "Condition" is Parse_Tree_Node condition

  else
    if Parse_Tree_Node connector = OR then
      Child = Parse_Tree_Node left_child
      Child from_node = Parse_Tree_Node from_node
      Child to_node = Parse_Tree_Node to_node
      Child = Parse_Tree_Node right_child                   ~ 238
      Child from_node = Parse_Tree_Node from_node
      Child to_node = Parse_Tree_Node to_node

  else  /* AND connector  */
      Next_Node = Next_Node_Id + 1
      Child = Parse_Tree_Node left_child
      Child from_node = Parse_Tree_Node from_node
      Child to_node = Next_Node_ID                          ~ 240
      Child = Parse_Tree_Node right_child
      Child to_node = Next_Node_ID
      Child to_node = Parse_Tree_Node to_node

  endif
 endif
enddo
```

FIG. 20



A<1 or (B<2 and C<3)

New_Node_Id = 3

	#condition		From Node	To Node
251	1	OR	1	2
252	2	A<1	–	–
253	3	AND	–	–
254	4	B<2	–	–
255	5	C<3	–	–

FIG. 21

Arc list

266

condition	From Node	To Node
A<1	1	2 260
B<2	1	3 262
C<3	2	3 264

Final arc list for: A<1 or (B<2 and C<3)

FIG. 22